RsPulseSeq

Release 2.7.1.30

Rohde & Schwarz

Oct 12, 2023

CONTENTS:


PULSESQ

CHAPTER

ONE

REVISION HISTORY

1.1 RsPulseSeq

Rohde & Schwarz Pulse Sequencer radar simulation software RsPulseSeq instrument driver.

Basic Hello-World code:

```
from RsPulseSeq import *

instr = RsPulseSeq('TCPIP::192.168.2.101::hislip0')
idn = instr.query('*IDN?')
print('Hello, I am: ' + idn)
```

Supported instruments: PulseSequencer

The package is hosted here: https://pypi.org/project/RsPulseSeq/

Documentation: https://RsPulseSeq.readthedocs.io/

Examples: https://github.com/Rohde-Schwarz/Examples/

1.1.1 Version history

Latest release notes summary: Update for SW version 2.7

Version 2.7.1

- Update for SW version 2.7

Version 2.4.1

- Included all three variants - RF, Digital, DFS

Version 2.4.0

- First release for FW 2.4

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsPulseSeq is a Python remote-control communication module for Rohde & Schwarz SCPI-based Test and Measurement Instruments. It represents SCPI commands as fixed APIs and hence provides SCPI autocompletion and helps you to avoid common string typing mistakes.

> Basic example of the idea:
> SCPI command:
> `SYSTem:REFerence:FREQuency:SOURce`
> Python module representation:
> writing:
> `driver.system.reference.frequency.source.set()`
> reading:
> `driver.system.reference.frequency.source.get()`

Check out this RsPulseSeq example:

```
"""Getting started - how to work with RsPulseSeq Python package.
This example performs basic RF settings on an R&S Pulse Sequence Software.
It shows the RsPulseSeq calls and their corresponding SCPI commands.
Notice that the python RsPulseSeq interfaces track the SCPI commands syntax."""

from RsPulseSeq import *

# Open the session
ps = RsPulseSeq('TCPIP::10.102.52.44::HISLIP', False, False)
# Greetings, stranger...
print(f'Hello, I am: {ps.utilities.idn_string}')

#   SOURce:FREQuency:FIXed 223000000
ps.source.frequency.cw.set_value(223E6)

ps.source.areGenerator.radar.base.set_attenuation(10)
```

(continues on next page)

(continued from previous page)

```
# Close the session
ps.close()
```

Couple of reasons why to choose this module over plain SCPI approach:

- Type-safe API using typing module
- You can still use the plain SCPI communication
- You can select which VISA to use or even not use any VISA at all
- Initialization of a new session is straight-forward, no need to set any other properties
- Many useful features are already implemented - reset, self-test, opc-synchronization, error checking, option checking
- Binary data blocks transfer in both directions
- Transfer of arrays of numbers in binary or ASCII format
- File transfers in both directions
- Events generation in case of error, sent data, received data, chunk data (for big files transfer)
- Multithreading session locking - you can use multiple threads talking to one instrument at the same time
- Logging feature tailored for SCPI communication - different for binary and ascii data

2.2 Installation

RsPulseSeq is hosted on pypi.org. You can install it with pip (for example, `pip.exe` for Windows), or if you are using Pycharm (and you should be :-) direct in the Pycharm `Packet Management GUI`.

Preconditions

- Installed VISA. You can skip this if you plan to use only socket LAN connection. Download the Rohde & Schwarz VISA for Windows, Linux, Mac OS from here

Option 1 - Installing with pip.exe under Windows

- Start the command console: `WinKey + R`, type `cmd` and hit ENTER
- Change the working directory to the Python installation of your choice (adjust the user name and python version in the path):

  ```
  cd c:\Users\John\AppData\Local\Programs\Python\Python37\Scripts
  ```

- Install with the command: `pip install RsPulseSeq`

Option 2 - Installing in Pycharm

- In Pycharm Menu `File->Settings->Project->Project Interpreter` click on the '+' button on the top left (the last PyCharm version)
- Type `RsPulseSeq` in the search box
- If you are behind a Proxy server, configure it in the Menu: `File->Settings->Appearance->System Settings->HTTP Proxy`

For more information about Rohde & Schwarz instrument remote control, check out our Instrument_Remote_Control_Web_Series .

Option 3 - Offline Installation

If you are still reading the installation chapter, it is probably because the options above did not work for you - proxy problems, your boss saw the internet bill... Here are 6 step for installing the RsPulseSeq offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsPulseSeq needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsPulseSeq package to your computer from the pypi.org: https://pypi.org/project/RsPulseSeq/#files to for example `c:\temp\`
- Start the command line `WinKey + R`, type `cmd` and hit ENTER
- Change the working directory to the Python installation of your choice (adjust the user name and python version in the path):

```
cd c:\Users\John\AppData\Local\Programs\Python\Python37\Scripts
```

- Install with the command: `pip install c:\temp\RsPulseSeq-2.7.1.30.tar`

2.3 Finding Available Instruments

Like the pyvisa's ResourceManager, the RsPulseSeq can search for available instruments:

```
"""
Find the instruments in your environment
"""

from RsPulseSeq import *

# Use the instr_list string items as resource names in the RsPulseSeq constructor
instr_list = RsPulseSeq.list_resources("?*")
print(instr_list)
```

If you have more VISAs installed, the one actually used by default is defined by a secret widget called Visa Conflict Manager. You can force your program to use a VISA of your choice:

```
"""
Find the instruments in your environment with the defined VISA implementation
"""
```

(continues on next page)

(continued from previous page)

```
from RsPulseSeq import *

# In the optional parameter visa_select you can use for example 'rs' or 'ni'
# Rs Visa also finds any NRP-Zxx USB sensors
instr_list = RsPulseSeq.list_resources('?*', 'rs')
print(instr_list)
```

Tip: We believe our R&S VISA is the best choice for our customers. Here are the reasons why:

- Small footprint
- Superior VXI-11 and HiSLIP performance
- Integrated legacy sensors NRP-Zxx support
- Additional VXI-11 and LXI devices search
- Availability for Windows, Linux, Mac OS

2.4 Initiating Instrument Session

RsPulseSeq offers four different types of starting your remote-control session. We begin with the most typical case, and progress with more special ones.

Standard Session Initialization

Initiating new instrument session happens, when you instantiate the RsPulseSeq object. Below, is a simple Hello World example. Different resource names are examples for different physical interfaces.

```
"""
Simple example on how to use the RsPulseSeq module for remote-controlling your instrument
Preconditions:

- Installed RsPulseSeq Python module Version 2.7.1 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsPulseSeq import *

# A good practice is to assure that you have a certain minimum version installed
RsPulseSeq.assert_minimum_version('2.7.1')
resource_string_1 = 'TCPIP::192.168.2.101::INSTR'  # Standard LAN connection (also␣
↪called VXI-11)
resource_string_2 = 'TCPIP::192.168.2.101::hislip0'  # Hi-Speed LAN connection - see␣
↪1MA208
resource_string_3 = 'GPIB::20::INSTR'  # GPIB Connection
resource_string_4 = 'USB::0x0AAD::0x0119::022019943::INSTR'  # USB-TMC (Test and␣
↪Measurement Class)

# Initializing the session
```

(continues on next page)

(continued from previous page)

```
driver = RsPulseSeq(resource_string_1)

idn = driver.utilities.query_str('*IDN?')
print(f"\nHello, I am: '{idn}'")
print(f'RsPulseSeq package version: {driver.utilities.driver_version}')
print(f'Visa manufacturer: {driver.utilities.visa_manufacturer}')
print(f'Instrument full name: {driver.utilities.full_instrument_model_name}')
print(f'Instrument installed options: {",".join(driver.utilities.instrument_options)}')

# Close the session
driver.close()
```

Note: If you are wondering about the missing `ASRL1::INSTR`, yes, it works too, but come on… it's 2023.

Do not care about specialty of each session kind; RsPulseSeq handles all the necessary session settings for you. You immediately have access to many identification properties in the interface `driver.utilities` . Here are same of them:

- `idn_string`
- `driver_version`
- `visa_manufacturer`
- `full_instrument_model_name`
- `instrument_serial_number`
- `instrument_firmware_version`
- `instrument_options`

The constructor also contains optional boolean arguments `id_query` and `reset`:

```
driver = RsPulseSeq('TCPIP::192.168.56.101::hislip0', id_query=True, reset=True)
```

- Setting `id_query` to True (default is True) checks, whether your instrument can be used with the RsPulseSeq module.
- Setting `reset` to True (default is False) resets your instrument. It is equivalent to calling the `reset()` method.

Selecting a Specific VISA

Just like in the function `list_resources()`, the RsPulseSeq allows you to choose which VISA to use:

```
"""
Choosing VISA implementation
"""

from RsPulseSeq import *

# Force use of the Rs Visa. For NI Visa, use the "SelectVisa='ni'"
driver = RsPulseSeq('TCPIP::192.168.56.101::INSTR', True, True, "SelectVisa='rs'")

idn = driver.utilities.query_str('*IDN?')
```

(continues on next page)

(continued from previous page)

```
print(f"\nHello, I am: '{idn}'")
print(f"\nI am using the VISA from: {driver.utilities.visa_manufacturer}")

# Close the session
driver.close()
```

No VISA Session

We recommend using VISA when possible preferrably with HiSlip session because of its low latency. However, if you are a strict VISA denier, RsPulseSeq has something for you too - **no Visa installation raw LAN socket**:

```
"""
Using RsPulseSeq without VISA for LAN Raw socket communication
"""

from RsPulseSeq import *

driver = RsPulseSeq('TCPIP::192.168.56.101::5025::SOCKET', True, True, "SelectVisa=
↪'socket'")
print(f'Visa manufacturer: {driver.utilities.visa_manufacturer}')
print(f"\nHello, I am: '{driver.utilities.idn_string}'")

# Close the session
driver.close()
```

Warning: Not using VISA can cause problems by debugging when you want to use the communication Trace Tool. The good news is, you can easily switch to use VISA and back just by changing the constructor arguments. The rest of your code stays unchanged.

Simulating Session

If a colleague is currently occupying your instrument, leave him in peace, and open a simulating session:

```
driver = RsPulseSeq('TCPIP::192.168.56.101::hislip0', True, True, "Simulate=True")
```

More `option_string` tokens are separated by comma:

```
driver = RsPulseSeq('TCPIP::192.168.56.101::hislip0', True, True, "SelectVisa='rs',␣
↪Simulate=True")
```

Shared Session

In some scenarios, you want to have two independent objects talking to the same instrument. Rather than opening a second VISA connection, share the same one between two or more RsPulseSeq objects:

```
"""
Sharing the same physical VISA session by two different RsPulseSeq objects
"""

from RsPulseSeq import *

driver1 = RsPulseSeq('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsPulseSeq.from_existing_session(driver1)

print(f'driver1: {driver1.utilities.idn_string}')
print(f'driver2: {driver2.utilities.idn_string}')

# Closing the driver2 session does not close the driver1 session - driver1 is the
↪'session master'
driver2.close()
print(f'driver2: I am closed now')

print(f'driver1: I am  still opened and working: {driver1.utilities.idn_string}')
driver1.close()
print(f'driver1: Only now I am closed.')
```

Note: The `driver1` is the object holding the 'master' session. If you call the `driver1.close()`, the `driver2` loses its instrument session as well, and becomes pretty much useless.

2.5 Plain SCPI Communication

After you have opened the session, you can use the instrument-specific part described in the RsPulseSeq API Structure. If for any reason you want to use the plain SCPI, use the `utilities` interface's two basic methods:

- `write_str()` - writing a command without an answer, for example ***RST**
- `query_str()` - querying your instrument, for example the ***IDN?** query

You may ask a question. Actually, two questions:

- **Q1**: Why there are not called `write()` and `query()` ?
- **Q2**: Where is the `read()` ?

Answer 1: Actually, there are - the `write_str()` / `write()` and `query_str()` / `query()` are aliases, and you can use any of them. We promote the `_str` names, to clearly show you want to work with strings. Strings in Python3 are Unicode, the *bytes* and *string* objects are not interchangeable, since one character might be represented by more than 1 byte. To avoid mixing string and binary communication, all the method names for binary transfer contain `_bin` in the name.

Answer 2: Short answer - you do not need it. Long answer - your instrument never sends unsolicited responses. If you send a set command, you use `write_str()`. For a query command, you use `query_str()`. So, you really do not need it...

Bottom line - if you are used to `write()` and `query()` methods, from pyvisa, the `write_str()` and `query_str()` are their equivalents.

Enough with the theory, let us look at an example. Simple write, and query:

```
"""
Basic string write_str / query_str
"""

from RsPulseSeq import *

driver = RsPulseSeq('TCPIP::192.168.56.101::INSTR')
driver.utilities.write_str('*RST')
response = driver.utilities.query_str('*IDN?')
print(response)

# Close the session
driver.close()
```

This example is so-called "*University-Professor-Example*" - good to show a principle, but never used in praxis. The abovementioned commands are already a part of the driver's API. Here is another example, achieving the same goal:

```
"""
Basic string write_str / query_str
"""

from RsPulseSeq import *

driver = RsPulseSeq('TCPIP::192.168.56.101::INSTR')
driver.utilities.reset()
print(driver.utilities.idn_string)

# Close the session
driver.close()
```

One additional feature we need to mention here: **VISA timeout**. To simplify, VISA timeout plays a role in each `query_xxx()`, where the controller (your PC) has to prevent waiting forever for an answer from your instrument. VISA timeout defines that maximum waiting time. You can set/read it with the `visa_timeout` property:

```
# Timeout in milliseconds
driver.utilities.visa_timeout = 3000
```

After this time, the RsPulseSeq raises an exception. Speaking of exceptions, an important feature of the RsPulseSeq is **Instrument Status Checking**. Check out the next chapter that describes the error checking in details.

For completion, we mention other string-based `write_xxx()` and `query_xxx()` methods - all in one example. They are convenient extensions providing type-safe float/boolean/integer setting/querying features:

```
"""
Basic string write_xxx / query_xxx
"""

from RsPulseSeq import *

driver = RsPulseSeq('TCPIP::192.168.56.101::INSTR')
```

(continues on next page)

(continued from previous page)

```
driver.utilities.visa_timeout = 5000
driver.utilities.instrument_status_checking = True
driver.utilities.write_int('SWEEP:COUNT ', 10)  # sending 'SWEEP:COUNT 10'
driver.utilities.write_bool('SOURCE:RF:OUTPUT:STATE ', True)  # sending
↪'SOURCE:RF:OUTPUT:STATE ON'
driver.utilities.write_float('SOURCE:RF:FREQUENCY ', 1E9)  # sending 'SOURCE:RF:FREQUENCY␣
↪1000000000'

sc = driver.utilities.query_int('SWEEP:COUNT?')  # returning integer number sc=10
out = driver.utilities.query_bool('SOURCE:RF:OUTPUT:STATE?')  # returning boolean␣
↪out=True
freq = driver.utilities.query_float('SOURCE:RF:FREQUENCY?')  # returning float number␣
↪freq=1E9

# Close the session
driver.close()
```

Lastly, a method providing basic synchronization: `query_opc()`. It sends query ***OPC?** to your instrument. The instrument waits with the answer until all the tasks it currently has in a queue are finished. This way your program waits too, and this way it is synchronized with the actions in the instrument. Remember to have the VISA timeout set to an appropriate value to prevent the timeout exception. Here's the snippet:

```
driver.utilities.visa_timeout = 3000
driver.utilities.write_str("INIT")
driver.utilities.query_opc()

# The results are ready now to fetch
results = driver.utilities.query_str("FETCH:MEASUREMENT?")
```

Tip: Wait, there's more: you can send the ***OPC?** after each `write_xxx()` automatically:

```
# Default value after init is False
driver.utilities.opc_query_after_write = True
```

2.6 Error Checking

RsPulseSeq pushes limits even further (internal R&S joke): It has a built-in mechanism that after each command/query checks the instrument's status subsystem, and raises an exception if it detects an error. For those who are already screaming: **Speed Performance Penalty!!!**, don't worry, you can disable it.

Instrument status checking is very useful since in case your command/query caused an error, you are immediately informed about it. Status checking has in most cases no practical effect on the speed performance of your program. However, if for example, you do many repetitions of short write/query sequences, it might make a difference to switch it off:

```
# Default value after init is True
driver.utilities.instrument_status_checking = False
```

To clear the instrument status subsystem of all errors, call this method:

```
driver.utilities.clear_status()
```

Instrument's status system error queue is clear-on-read. It means, if you query its content, you clear it at the same time. To query and clear list of all the current errors, use this snippet:

```
errors_list = driver.utilities.query_all_errors()
```

See the next chapter on how to react on errors.

2.7 Exception Handling

The base class for all the exceptions raised by the RsPulseSeq is `RsInstrException`. Inherited exception classes:

- `ResourceError` raised in the constructor by problems with initiating the instrument, for example wrong or non-existing resource name
- `StatusException` raised if a command or a query generated error in the instrument's error queue
- `TimeoutException` raised if a visa timeout or an opc timeout is reached

In this example we show usage of all of them. Because it is difficult to generate an error using the instrument-specific SCPI API, we use plain SCPI commands:

```
"""
Showing how to deal with exceptions
"""

from RsPulseSeq import *

driver = None
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsPulseSeq('TCPIP::10.112.1.179::hislip0')
except ResourceError as e:
    print(e.args[0])
    print('Your instrument is probably OFF...')
    # Exit now, no point of continuing
    exit(1)

# Dealing with commands that potentially generate errors OPTION 1:
# Switching the status checking OFF termporarily
driver.utilities.instrument_status_checking = False
driver.utilities.write_str('MY:MISSpelled:COMMand')
# Clear the error queue
driver.utilities.clear_status()
# Status checking ON again
driver.utilities.instrument_status_checking = True

# Dealing with queries that potentially generate errors OPTION 2:
try:
    # You migh want to reduce the VISA timeout to avoid long waiting
    driver.utilities.visa_timeout = 1000
    driver.utilities.query_str('MY:WRONg:QUERy?')
```

(continues on next page)

(continued from previous page)

```
except StatusException as e:
    # Instrument status error
    print(e.args[0])
    print('Nothing to see here, moving on...')

except TimeoutException as e:
    # Timeout error
    print(e.args[0])
    print('That took a long time...')

except RsInstrException as e:
    # RsInstrException is a base class for all the RsPulseSeq exceptions
    print(e.args[0])
    print('Some other RsPulseSeq error...')

finally:
    driver.utilities.visa_timeout = 5000
    # Close the session in any case
    driver.close()
```

Tip: General rules for exception handling:

- If you are sending commands that might generate errors in the instrument, for example deleting a file which does not exist, use the **OPTION 1** - temporarily disable status checking, send the command, clear the error queue and enable the status checking again.
- If you are sending queries that might generate errors or timeouts, for example querying measurement that can not be performed at the moment, use the **OPTION 2** - try/except with optionally adjusting the timeouts.

2.8 Transferring Files

Instrument -> PC

You definitely experienced it: you just did a perfect measurement, saved the results as a screenshot to an instrument's storage drive. Now you want to transfer it to your PC. With RsPulseSeq, no problem, just figure out where the screenshot was stored on the instrument. In our case, it is */var/user/instr_screenshot.png*:

```
driver.utilities.read_file_from_instrument_to_pc(
    r'/var/user/instr_screenshot.png',
    r'c:\temp\pc_screenshot.png')
```

PC -> Instrument

Another common scenario: Your cool test program contains a setup file you want to transfer to your instrument: Here is the RsPulseSeq one-liner split into 3 lines:

```
driver.utilities.send_file_from_pc_to_instrument(
    r'c:\MyCoolTestProgram\instr_setup.sav',
    r'/var/appdata/instr_setup.sav')
```

2.9 Writing Binary Data

Writing from bytes

An example where you need to send binary data is a waveform file of a vector signal generator. First, you compose your wform_data as `bytes`, and then you send it with `write_bin_block()`:

```
# MyWaveform.wv is an instrument file name under which this data is stored
driver.utilities.write_bin_block(
    "SOUR:BB:ARB:WAV:DATA 'MyWaveform.wv',",
    wform_data)
```

Note: Notice the `write_bin_block()` has two parameters:

- `string` parameter `cmd` for the SCPI command
- `bytes` parameter `payload` for the actual binary data to send

Writing from PC files

Similar to querying binary data to a file, you can write binary data from a file. The second parameter is then the PC file path the content of which you want to send:

```
driver.utilities.write_bin_block_from_file(
    "SOUR:BB:ARB:WAV:DATA 'MyWaveform.wv',",
    r"c:\temp\wform_data.wv")
```

2.10 Transferring Big Data with Progress

We can agree that it can be annoying using an application that shows no progress for long-lasting operations. The same is true for remote-control programs. Luckily, the RsPulseSeq has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

RsPulseSeq allows you to register a function (programmers fancy name is `callback`), which is then periodicaly invoked after transfer of one data chunk. You can define that chunk size, which gives you control over the callback invoke frequency. You can even slow down the transfer speed, if you want to process the data as they arrive (direction instrument -> PC).

To show this in praxis, we are going to use another *University-Professor-Example*: querying the ***IDN?** with chunk size of 2 bytes and delay of 200ms between each chunk read:

```
"""
Event handlers by reading
"""

from RsPulseSeq import *
import time


def my_transfer_handler(args):
    """Function called each time a chunk of data is transferred"""
    # Total size is not always known at the beginning of the transfer
    total_size = args.total_size if args.total_size is not None else "unknown"

    print(f"Context: '{args.context}{'with opc' if args.opc_sync else ''}', "
        f"chunk {args.chunk_ix}, "
        f"transferred {args.transferred_size} bytes, "
        f"total size {total_size}, "
        f"direction {'reading' if args.reading else 'writing'}, "
        f"data '{args.data}'")

    if args.end_of_transfer:
        print('End of Transfer')
    time.sleep(0.2)


driver = RsPulseSeq('TCPIP::192.168.56.101::INSTR')

driver.events.on_read_handler = my_transfer_handler
# Switch on the data to be included in the event arguments
# The event arguments args.data will be updated
driver.events.io_events_include_data = True
# Set data chunk size to 2 bytes
driver.utilities.data_chunk_size = 2
driver.utilities.query_str('*IDN?')
# Unregister the event handler
driver.utilities.on_read_handler = None

# Close the session
driver.close()
```

If you start it, you might wonder (or maybe not): why is the `args.total_size = None`? The reason is, in this particular case the RsPulseSeq does not know the size of the complete response up-front. However, if you use the same mechanism for transfer of a known data size (for example, file transfer), you get the information about the total size too, and hence you can calculate the progress as:

*progress [pct] = 100 * args.transferred_size / args.total_size*

Snippet of transferring file from PC to instrument, the rest of the code is the same as in the previous example:

```
driver.events.on_write_handler = my_transfer_handler
driver.events.io_events_include_data = True
driver.data_chunk_size = 1000
driver.utilities.send_file_from_pc_to_instrument(
    r'c:\MyCoolTestProgram\my_big_file.bin',
```

(continues on next page)

(continued from previous page)

```
    r'/var/user/my_big_file.bin')
# Unregister the event handler
driver.events.on_write_handler = None
```

2.11 Multithreading

You are at the party, many people talking over each other. Not every person can deal with such crosstalk, neither can measurement instruments. For this reason, RsPulseSeq has a feature of scheduling the access to your instrument by using so-called **Locks**. Locks make sure that there can be just one client at a time *talking* to your instrument. Talking in this context means completing one communication step - one command write or write/read or write/read/error check.

To describe how it works, and where it matters, we take three typical mulithread scenarios:

One instrument session, accessed from multiple threads

You are all set - the lock is a part of your instrument session. Check out the following example - it will execute properly, although the instrument gets 10 queries at the same time:

```
"""
Multiple threads are accessing one RsPulseSeq object
"""

import threading
from RsPulseSeq import *


def execute(session):
    """Executed in a separate thread."""
    session.utilities.query_str('*IDN?')


driver = RsPulseSeq('TCPIP::192.168.56.101::INSTR')
threads = []
for i in range(10):
    t = threading.Thread(target=execute, args=(driver, ))
    t.start()
    threads.append(t)
print('All threads started')

# Wait for all threads to join this main thread
for t in threads:
    t.join()
print('All threads ended')

driver.close()
```

Shared instrument session, accessed from multiple threads

Same as the previous case, you are all set. The session carries the lock with it. You have two objects, talking to the same instrument from multiple threads. Since the instrument session is shared, the same lock applies to both objects causing the exclusive access to the instrument.

Try the following example:

```
"""
Multiple threads are accessing two RsPulseSeq objects with shared session
"""

import threading
from RsPulseSeq import *


def execute(session: RsPulseSeq, session_ix, index) -> None:
    """Executed in a separate thread."""
    print(f'{index} session {session_ix} query start...')
    session.utilities.query_str('*IDN?')
    print(f'{index} session {session_ix} query end')


driver1 = RsPulseSeq('TCPIP::192.168.56.101::INSTR')
driver2 = RsPulseSeq.from_existing_session(driver1)
driver1.utilities.visa_timeout = 200
driver2.utilities.visa_timeout = 200
# To see the effect of crosstalk, uncomment this line
# driver2.utilities.clear_lock()

threads = []
for i in range(10):
    t = threading.Thread(target=execute, args=(driver1, 1, i,))
    t.start()
    threads.append(t)
    t = threading.Thread(target=execute, args=(driver2, 2, i,))
    t.start()
    threads.append(t)
print('All threads started')

# Wait for all threads to join this main thread
for t in threads:
    t.join()
print('All threads ended')

driver2.close()
driver1.close()
```

As you see, everything works fine. If you want to simulate some party crosstalk, uncomment the line `driver2.utilities.clear_lock()`. Thich causes the driver2 session lock to break away from the driver1 session lock. Although the driver1 still tries to schedule its instrument access, the driver2 tries to do the same at the same time, which leads to all the fun stuff happening.

Multiple instrument sessions accessed from multiple threads

Here, there are two possible scenarios depending on the instrument's VISA interface:

- Your are lucky, because you instrument handles each remote session completely separately. An example of such instrument is SMW200A. In this case, you have no need for session locking.
- Your instrument handles all sessions with one set of in/out buffers. You need to lock the session for the duration of a talk. And you are lucky again, because the RsPulseSeq takes care of it for you. The text below describes this scenario.

Run the following example:

```
"""
Multiple threads are accessing two RsPulseSeq objects with two separate sessions
"""

import threading
from RsPulseSeq import *


def execute(session: RsPulseSeq, session_ix, index) -> None:
    """Executed in a separate thread."""
    print(f'{index} session {session_ix} query start...')
    session.utilities.query_str('*IDN?')
    print(f'{index} session {session_ix} query end')


driver1 = RsPulseSeq('TCPIP::192.168.56.101::INSTR')
driver2 = RsPulseSeq('TCPIP::192.168.56.101::INSTR')
driver1.utilities.visa_timeout = 200
driver2.utilities.visa_timeout = 200

# Synchronise the sessions by sharing the same lock
driver2.utilities.assign_lock(driver1.utilities.get_lock())  # To see the effect of␣
↪crosstalk, comment this line

threads = []
for i in range(10):
    t = threading.Thread(target=execute, args=(driver1, 1, i,))
    t.start()
    threads.append(t)
    t = threading.Thread(target=execute, args=(driver2, 2, i,))
    t.start()
    threads.append(t)
print('All threads started')

# Wait for all threads to join this main thread
for t in threads:
    t.join()
print('All threads ended')

driver2.close()
driver1.close()
```

You have two completely independent sessions that want to talk to the same instrument at the same time. This will

not go well, unless they share the same session lock. The key command to achieve this is `driver2.utilities.assign_lock(driver1.utilities.get_lock())` Try to comment it and see how it goes. If despite commenting the line the example runs without issues, you are lucky to have an instrument similar to the SMW200A.

2.12 Logging

Yes, the logging again. This one is tailored for instrument communication. You will appreciate such handy feature when you troubleshoot your program, or just want to protocol the SCPI communication for your test reports.

What can you actually do with the logger?

- Write SCPI communication to a stream-like object, for example console or file, or both simultaneously
- Log only errors and skip problem-free parts; this way you avoid going through thousands lines of texts
- Investigate duration of certain operations to optimize your program's performance
- Log custom messages from your program

Let us take this basic example:

```
"""
Basic logging example to the console
"""

from RsPulseSeq import *

driver = RsPulseSeq('TCPIP::192.168.1.101::INSTR')

# Switch ON logging to the console.
driver.utilities.logger.log_to_console = True
driver.utilities.logger.mode = LoggingMode.On
driver.utilities.reset()

# Close the session
driver.close()
```

Console output:

```
10:29:10.819     TCPIP::192.168.1.101::INSTR     0.976 ms  Write: *RST
10:29:10.819     TCPIP::192.168.1.101::INSTR  1884.985 ms  Status check: OK
10:29:12.704     TCPIP::192.168.1.101::INSTR     0.983 ms  Query OPC: 1
10:29:12.705     TCPIP::192.168.1.101::INSTR     2.892 ms  Clear status: OK
10:29:12.708     TCPIP::192.168.1.101::INSTR     3.905 ms  Status check: OK
10:29:12.712     TCPIP::192.168.1.101::INSTR     1.952 ms  Close: Closing session
```

The columns of the log are aligned for better reading. Columns meaning:

- (1) Start time of the operation
- (2) Device resource name (you can set an alias)
- (3) Duration of the operation
- (4) Log entry

Tip: You can customize the logging format with `set_format_string()`, and set the maximum log entry length with the properties:

- `abbreviated_max_len_ascii`
- `abbreviated_max_len_bin`
- `abbreviated_max_len_list`

See the full logger help *here*.

Notice the SCPI communication starts from the line `driver.utilities.reset()`. If you want to log the initialization of the session as well, you have to switch the logging ON already in the constructor:

```
driver = RsPulseSeq('TCPIP::192.168.56.101::hislip0', options='LoggingMode=On')
```

Parallel to the console logging, you can log to a general stream. Do not fear the programmer's jargon'... under the term **stream** you can just imagine a file. To be a little more technical, a stream in Python is any object that has two methods: `write()` and `flush()`. This example opens a file and sets it as logging target:

```
"""
Example of logging to a file
"""

from RsPulseSeq import *

driver = RsPulseSeq('TCPIP::192.168.1.101::INSTR')

# We also want to log to the console.
driver.utilities.logger.log_to_console = True

# Logging target is our file
file = open(r'c:\temp\my_file.txt', 'w')
driver.utilities.logger.set_logging_target(file)
driver.utilities.logger.mode = LoggingMode.On

# Instead of the 'TCPIP::192.168.1.101::INSTR', show 'MyDevice'
driver.utilities.logger.device_name = 'MyDevice'

# Custom user entry
driver.utilities.logger.info_raw('----- This is my custom log entry. ---- ')

driver.utilities.reset()

# Close the session
driver.close()

# Close the log file
file.close()
```

Tip: To make the log more compact, you can skip all the lines with `Status check: OK`:

```
driver.utilities.logger.log_status_check_ok = False
```

Hint: You can share the logging file between multiple sessions. In such case, remember to close the file only after you have stopped logging in all your sessions, otherwise you get a log write error.

For logging to a UDP port in addition to other log targets, use one of the lines:

```
driver.utilities.logger.log_to_udp = True
driver.utilities.logger.log_to_console_and_udp = True
```

You can select the UDP port to log to, the default is 49200:

```
driver.utilities.logger.udp_port = 49200
```

Another cool feature is logging only errors. To make this mode usefull for troubleshooting, you also want to see the circumstances which lead to the errors. Each driver elementary operation, for example, `write_str()`, can generate a group of log entries - let us call them **Segment**. In the logging mode `Errors`, a whole segment is logged only if at least one entry of the segment is an error.

The script below demonstrates this feature. We use a direct SCPI communication to send a misspelled SCPI command *CLS, which leads to instrument status error:

```
"""
Logging example to the console with only errors logged
"""

from RsPulseSeq import *

driver = RsPulseSeq('TCPIP::192.168.1.101::INSTR', options='LoggingMode=Errors')

# Switch ON logging to the console.
driver.utilities.logger.log_to_console = True

# Reset will not be logged, since no error occurred there
driver.utilities.reset()

# Now a misspelled command.
driver.utilities.write('*CLaS')

# A good command again, no logging here
idn = driver.utilities.query('*IDN?')

# Close the session
driver.close()
```

Console output:

```
12:11:02.879 TCPIP::192.168.1.101::INSTR     0.976 ms  Write string: *CLaS
12:11:02.879 TCPIP::192.168.1.101::INSTR     6.833 ms  Status check: StatusException:
                                                       Instrument error detected: Undefined header;
↪*CLaS
```

Notice the following:

- Although the operation **Write string: *CLaS** finished without an error, it is still logged, because it provides the context for the actual error which occurred during the status checking right after.
- No other log entries are present, including the session initialization and close, because they were all error-free.

CHAPTER

THREE

ENUMS

3.1 AmType

```
# Example value:
value = enums.AmType.LSB
# All values (4x):
LSB | SB | STD | USB
```

3.2 AntennaModel

```
# First value:
value = enums.AntennaModel.ARRay
# Last value:
value = enums.AntennaModel.USER
# All values (12x):
ARRay | CARDoid | CARRay | COSecant | CUSTom | DIPole | GAUSsian | HORN
PARabolic | PLUGin | SINC | USER
```

3.3 AntennaModelArray

```
# Example value:
value = enums.AntennaModelArray.COSine
# All values (8x):
COSine | COSN | CSQuared | HAMMing | HANN | PARabolic | TRIangular | UNIForm
```

3.4 Attitude

```
# Example value:
value = enums.Attitude.CONStant
# All values (3x):
CONStant | MOTion | WAYPoint
```

3.5 AutoManualMode

```
# Example value:
value = enums.AutoManualMode.AUTO
# All values (2x):
AUTO | MANual
```

3.6 Azimuth

```
# Example value:
value = enums.Azimuth.BEARing
# All values (2x):
BEARing | RX
```

3.7 BarkerCode

```
# First value:
value = enums.BarkerCode.R11
# Last value:
value = enums.BarkerCode.R7
# All values (9x):
R11 | R13 | R2A | R2B | R3 | R4A | R4B | R5
R7
```

3.8 BaseDomain

```
# Example value:
value = enums.BaseDomain.PULSe
# All values (2x):
PULSe | TIME
```

3.9 BaseDomainB

```
# Example value:
value = enums.BaseDomainB.LENGth
# All values (2x):
LENGth | TIME
```

3.10 BbSync

```
# Example value:
value = enums.BbSync.CTRigger
# All values (3x):
CTRigger | TRIGger | UNSYnc
```

3.11 BlockSize

```
# Example value:
value = enums.BlockSize._16K
# All values (6x):
_16K | _1M | _2M | _32K | _4M | _64K
```

3.12 BlType

```
# Example value:
value = enums.BlType.MIRRor
# All values (2x):
MIRRor | OMNidirect
```

3.13 BpskTtype

```
# Example value:
value = enums.BpskTtype.COSine
# All values (2x):
COSine | LINear
```

3.14 BpskType

```
# Example value:
value = enums.BpskType.CONStant
# All values (2x):
CONStant | STANdard
```

3.15 BufferSize

```
# Example value:
value = enums.BufferSize._128M
# All values (6x):
_128M | _16M | _1G | _256M | _512M | _64M
```

3.16 ChirpType

```
# Example value:
value = enums.ChirpType.DOWN
# All values (5x):
DOWN | PIECewise | SINE | TRIangular | UP
```

3.17 CircularMode

```
# Example value:
value = enums.CircularMode.RPM
# All values (2x):
RPM | SEC
```

3.18 Coding

```
# Example value:
value = enums.Coding.DGRay
# All values (4x):
DGRay | DIFFerential | GRAY | NONE
```

3.19 Complexity

```
# Example value:
value = enums.Complexity.DIRection
# All values (3x):
DIRection | EMITter | PTRain
```

3.20 Condition

```
# Example value:
value = enums.Condition.EQUal
# All values (4x):
EQUal | GREater | NOTequal | SMALler
```

3.21 DataMop

```
# First value:
value = enums.DataMop.AM
# Last value:
value = enums.DataMop.TFM
# All values (20x):
AM | ASK | BKR11 | BKR13 | BKR2a | BKR2b | BKR3 | BKR4a
BKR4b | BKR5 | BKR7 | CPH | CW | FM | FSK | LFM
NLFM | PLFM | PSK | TFM
```

3.22 DataUnit

```
# Example value:
value = enums.DataUnit.DB
# All values (3x):
DB | VOLTage | WATTs
```

3.23 DfType

```
# Example value:
value = enums.DfType.BACKground
# All values (4x):
BACKground | EMITter | PLATform | WAVeform
```

3.24 EnvelopeMode

```
# Example value:
value = enums.EnvelopeMode.DATA
# All values (2x):
DATA | EQUation
```

3.25 ExcMode

```
# Example value:
value = enums.ExcMode.LEVel
# All values (3x):
LEVel | TIME | WIDTh
```

3.26 FillerMode

```
# Example value:
value = enums.FillerMode.DURation
# All values (2x):
DURation | TSYNc
```

3.27 FillerSignal

```
# Example value:
value = enums.FillerSignal.BLANk
# All values (3x):
BLANk | CW | HOLD
```

3.28 FillerTime

```
# Example value:
value = enums.FillerTime.EQUation
# All values (2x):
EQUation | FIXed
```

3.29 FilterType

```
# First value:
value = enums.FilterType.COS
# Last value:
value = enums.FilterType.SOQPsk
# All values (9x):
COS | FSKGauss | GAUSs | LPASs | NONE | RCOS | RECTangular | SMWRect
SOQPsk
```

3.30 FskType

```
# Example value:
value = enums.FskType.FS16
# All values (6x):
FS16 | FS2 | FS32 | FS4 | FS64 | FS8
```

3.31 GeneratorType

```
# Example value:
value = enums.GeneratorType.SGT
# All values (8x):
SGT | SMBB | SMBV | SMJ | SMM | SMU | SMW | SW
```

3.32 Geometry

```
# Example value:
value = enums.Geometry.CIRCular
# All values (4x):
CIRCular | HEXagonal | LINear | RECTangular
```

3.33 HqMode

```
# Example value:
value = enums.HqMode.NORMal
# All values (2x):
NORMal | TABLe
```

3.34 InterleaveMode

```
# Example value:
value = enums.InterleaveMode.DROP
# All values (2x):
DROP | MERGe
```

3.35 Interpolation

```
# Example value:
value = enums.Interpolation.LINear
# All values (2x):
LINear | NONE
```

3.36 IpmMode

```
# Example value:
value = enums.IpmMode.INDividual
# All values (2x):
INDividual | SAME
```

3.37 IpmPlotView

```
# Example value:
value = enums.IpmPlotView.HISTogram
# All values (2x):
HISTogram | TIMeseries
```

3.38 IpmType

```
# First value:
value = enums.IpmType.BINomial
# Last value:
value = enums.IpmType.WAVeform
# All values (10x):
BINomial | EQUation | LIST | PLUGin | RANDom | RLISt | RSTep | SHAPe
STEPs | WAVeform
```

3.39 ItemPattern

```
# First value:
value = enums.ItemPattern.ALT
# Last value:
value = enums.ItemPattern.ZERO
# All values (12x):
ALT | ONE | R11 | R13 | R2A | R2B | R3 | R4A
R4B | R5 | R7 | ZERO
```

3.40 ItemType

```
# Example value:
value = enums.ItemType.FILLer
# All values (6x):
FILLer | LOOP | OVL | PULSe | SUBSequence | WAVeform
```

3.41 ItemTypeB

```
# Example value:
value = enums.ItemTypeB.PATTern
# All values (3x):
PATTern | PRBS | USER
```

3.42 Lattice

```
# Example value:
value = enums.Lattice.RECTangular
# All values (2x):
RECTangular | TRIangular
```

3.43 LobesCount

```
# Example value:
value = enums.LobesCount._2
# All values (2x):
_2 | _4
```

3.44 LoopType

```
# Example value:
value = enums.LoopType.FIXed
# All values (2x):
FIXed | VARiable
```

3.45 LswDirection

```
# Example value:
value = enums.LswDirection.H
# All values (2x):
H | V
```

3.46 ModuleType

```
# Example value:
value = enums.ModuleType.IPM
# All values (3x):
IPM | MOP | REPort
```

3.47 MopType

```
# First value:
value = enums.MopType.AM
# Last value:
value = enums.MopType.QPSK
# All values (21x):
AM | AMSTep | ASK | BARKer | BPSK | CCHiprp | CHIRp | FM
FMSTep | FSK | MSK | NLCHirp | NOISe | PCHirp | PLISt | PLUGin
POLYphase | PSK8 | PWISechirp | QAM | QPSK
```

3.48 MovementRframe

```
# Example value:
value = enums.MovementRframe.PZ
# All values (2x):
PZ | WGS
```

3.49 MovementRmode

```
# Example value:
value = enums.MovementRmode.CYCLic
# All values (3x):
CYCLic | ONEWay | ROUNdtrip
```

3.50 MovementType

```
# Example value:
value = enums.MovementType.ARC
# All values (4x):
ARC | LINE | TRACe | WAYPoint
```

3.51 OutFormat

```
# Example value:
value = enums.OutFormat.ESEQencing
# All values (3x):
ESEQencing | MSW | WV
```

3.52 PhaseMode

```
# Example value:
value = enums.PhaseMode.ABSolute
# All values (3x):
ABSolute | CONTinuous | MEMory
```

3.53 Pmode

```
# Example value:
value = enums.Pmode.MOVing
# All values (2x):
MOVing | STATic
```

3.54 PmodeLocation

```
# Example value:
value = enums.PmodeLocation.MOVing
# All values (3x):
MOVing | STATic | STEPs
```

3.55 PmodSource

```
# Example value:
value = enums.PmodSource.EXTernal
# All values (3x):
EXTernal | INTernal | OFF
```

3.56 PolarCut

```
# Example value:
value = enums.PolarCut.XY
# All values (2x):
XY | YZ
```

3.57 Polarization

```
# Example value:
value = enums.Polarization.CLEFt
# All values (6x):
CLEFt | CRIGht | HORizontal | SLEFt | SRIGht | VERTical
```

3.58 PolarType

```
# Example value:
value = enums.PolarType.CARTesian
# All values (2x):
CARTesian | POLar
```

3.59 PolynomType

```
# Example value:
value = enums.PolynomType.FRANk
# All values (5x):
FRANk | P1 | P2 | P3 | P4
```

3.60 PrbsType

```
# Example value:
value = enums.PrbsType.P11
# All values (8x):
P11 | P15 | P16 | P20 | P21 | P23 | P7 | P9
```

3.61 PreviewMode

```
# Example value:
value = enums.PreviewMode.ENVelope
# All values (2x):
ENVelope | MOP
```

3.62 PreviewMop

```
# Example value:
value = enums.PreviewMop.FREQuency
# All values (3x):
FREQuency | IQ | PHASe
```

3.63 ProgramMode

```
# Example value:
value = enums.ProgramMode.DEMO
# All values (3x):
DEMO | EXPert | STANdard
```

3.64 Psec

```
# First value:
value = enums.Psec.NONE
# Last value:
value = enums.Psec.SEC9
# All values (18x):
NONE | PRIMary | SEC1 | SEC10 | SEC11 | SEC12 | SEC13 | SEC14
SEC15 | SEC16 | SEC2 | SEC3 | SEC4 | SEC5 | SEC6 | SEC7
SEC8 | SEC9
```

3.65 PulseSetting

```
# Example value:
value = enums.PulseSetting.GENeral
# All values (5x):
GENeral | LEVel | MKR | MOP | TIMing
```

3.66 PulseType

```
# Example value:
value = enums.PulseType.COSine
# All values (4x):
COSine | LINear | RCOSine | SQRT
```

3.67 PwdType

```
# Example value:
value = enums.PwdType.AMMos
# All values (4x):
AMMos | DEFault | PLUGin | TEMPlate
```

3.68 QamType

```
# Example value:
value = enums.QamType.Q128
# All values (5x):
Q128 | Q16 | Q256 | Q32 | Q64
```

3.69 QpskType

```
# Example value:
value = enums.QpskType.ASOQpsk
# All values (6x):
ASOQpsk | BSOQpsk | DQPSk | NORMal | OQPSk | TGSoqpsk
```

3.70 RandomDistribution

```
# Example value:
value = enums.RandomDistribution.NORMal
# All values (3x):
NORMal | U | UNIForm
```

3.71 RasterDirection

```
# Example value:
value = enums.RasterDirection.HORizontal
# All values (2x):
HORizontal | VERTical
```

3.72 RecModel

```
# Example value:
value = enums.RecModel.COMBined
# All values (3x):
COMBined | INTerfero | TDOA
```

3.73 RepeatMode

```
# Example value:
value = enums.RepeatMode.CONTinuous
# All values (2x):
CONTinuous | SINGle
```

3.74 RepetitionType

```
# Example value:
value = enums.RepetitionType.DURation
# All values (3x):
DURation | FIXed | VARiable
```

3.75 Rotation

```
# Example value:
value = enums.Rotation.CCW
# All values (2x):
CCW | CW
```

3.76 SanitizeScenario

```
# Example value:
value = enums.SanitizeScenario.ALL
# All values (3x):
ALL | REPository | SCENario
```

3.77 ScanType

```
# First value:
value = enums.ScanType.CIRCular
# Last value:
value = enums.ScanType.SPIRal
# All values (10x):
CIRCular | CONical | CUSTom | HELical | LISSajous | LSW | RASTer | SECTor
SIN | SPIRal
```

3.78 ScenarioType

```
# First value:
value = enums.ScenarioType.CEMitter
# Last value:
value = enums.ScenarioType.WAVeform
# All values (9x):
CEMitter | CSEQuence | DF | DYNamic | EMITter | LOCalized | PDW | SEQuence
WAVeform
```

3.79 SecurityLevel

```
# Example value:
value = enums.SecurityLevel.LEV0
# All values (5x):
LEV0 | LEV1 | LEV2 | LEV3 | LEV4
```

3.80 SequenceType

```
# Example value:
value = enums.SequenceType.PULSe
# All values (2x):
PULSe | WAVeform
```

3.81 SigCont

```
# Example value:
value = enums.SigCont.COMM
# All values (2x):
COMM | PULSe
```

3.82 SourceInt

```
# Example value:
value = enums.SourceInt.EXTernal
# All values (2x):
EXTernal | INTernal
```

3.83 SourceType

```
# Example value:
value = enums.SourceType.PROFile
# All values (2x):
PROFile | VARiable
```

3.84 State

```
# Example value:
value = enums.State.IDLE
# All values (2x):
IDLE | RUN
```

3.85 TargetOut

```
# Example value:
value = enums.TargetOut.FILE
# All values (2x):
FILE | INSTrument
```

3.86 TargetParam

```
# First value:
value = enums.TargetParam.AMDepth
# Last value:
value = enums.TargetParam.WIDTh
# All values (20x):
AMDepth | AMFRequency | CDEViation | DELay | DROop | FALL | FMDeviation | FMFRequency
FREQuency | FSKDeviation | LEVel | OVERshoot | PHASe | PRF | PRI | RFRequency
RISE | RLEVel | SRATe | WIDTh
```

3.87 TargetType

```
# Example value:
value = enums.TargetType.PARameter
# All values (2x):
PARameter | VARiable
```

3.88 TimeMode

```
# Example value:
value = enums.TimeMode.PRF
# All values (2x):
PRF | PRI
```

3.89 TimeReference

```
# Example value:
value = enums.TimeReference.FULL
# All values (3x):
FULL | POWer | VOLTage
```

3.90 Units

```
# Example value:
value = enums.Units.DB
# All values (6x):
DB | DEGRees | HERTz | NONE | PERCent | SEConds
```

3.91 Vehicle

```
# Example value:
value = enums.Vehicle.AIRPlane
# All values (5x):
AIRPlane | LVEHicle | RECeiver | SHIP | STATionary
```

3.92 VehicleMovement

```
# Example value:
value = enums.VehicleMovement.AIRPlane
# All values (6x):
AIRPlane | CAR | DEFault | LVEHicle | SHIP | STATionary
```

3.93 ViewCount

```
# Example value:
value = enums.ViewCount._100
# All values (8x):
_100 | _1000 | _10000 | _100000 | _50 | _500 | _5000 | _50000
```

3.94 ViewXode

```
# Example value:
value = enums.ViewXode.SAMPles
# All values (2x):
SAMPles | TIME
```

3.95 WaveformShape

```
# Example value:
value = enums.WaveformShape.RAMP
# All values (3x):
RAMP | SINE | TRIangular
```

3.96 WaveformType

```
# First value:
value = enums.WaveformType.AIF
# Last value:
value = enums.WaveformType.WAVeform
# All values (10x):
AIF | APDW | BEMitter | CW | IQDW | MT | NOISe | PDW
USER | WAVeform
```

3.97 Ymode

```
# Example value:
value = enums.Ymode.FREQuency
# All values (7x):
FREQuency | IQ | MAGDb | MAGV | MAGW | PAV | PHASe
```

CHAPTER

FOUR

EXAMPLES

For more examples, visit our Rohde & Schwarz Github repository.

```
"""Getting started - how to work with RsPulseSeq Python package.
This example performs basic RF settings on an R&S Pulse Sequence Software.
It shows the RsPulseSeq calls and their corresponding SCPI commands.
Notice that the python RsPulseSeq interfaces track the SCPI commands syntax."""

from RsPulseSeq import *

# Open the session
ps = RsPulseSeq('TCPIP::10.102.52.44::HISLIP', False, False)
# Greetings, stranger...
print(f'Hello, I am: {ps.utilities.idn_string}')

#   SOURce:FREQuency:FIXed 223000000
ps.source.frequency.cw.set_value(223E6)

ps.source.areGenerator.radar.base.set_attenuation(10)

# Close the session
ps.close()
```

CHAPTER

FIVE

RSPULSESEQ API STRUCTURE

class RsPulseSeq(*resource_name: str*, *id_query: bool = True*, *reset: bool = False*, *options: str = None*, *direct_session: object = None*)

1297 total commands, 32 Subgroups, 0 group commands

Initializes new RsPulseSeq session.

Parameter options tokens examples:

- `Simulate=True` - starts the session in simulation mode. Default: `False`
- `SelectVisa=socket` - uses no VISA implementation for socket connections - you do not need any VISA-C installation
- `SelectVisa=rs` - forces usage of RohdeSchwarz Visa
- `SelectVisa=ivi` - forces usage of National Instruments Visa
- `QueryInstrumentStatus = False` - same as `driver.utilities.instrument_status_checking = False`. Default: `True`
- `WriteDelay = 20, ReadDelay = 5` - Introduces delay of 20ms before each write and 5ms before each read. Default: `0ms` for both
- `OpcWaitMode = OpcQuery` - mode for all the opc-synchronised write/reads. Other modes: StbPolling, StbPollingSlow, StbPollingSuperSlow. Default: `StbPolling`
- `AddTermCharToWriteBinBLock = True` - Adds one additional LF to the end of the binary data (some instruments require that). Default: `False`
- `AssureWriteWithTermChar = True` - Makes sure each command/query is terminated with termination character. Default: Interface dependent
- `TerminationCharacter = "\r"` - Sets the termination character for reading. Default: `\n` (LineFeed or LF)
- `DataChunkSize = 10E3` - Maximum size of one write/read segment. If transferred data is bigger, it is split to more segments. Default: `1E6` bytes
- `OpcTimeout = 10000` - same as driver.utilities.opc_timeout = 10000. Default: `30000ms`
- `VisaTimeout = 5000` - same as driver.utilities.visa_timeout = 5000. Default: `10000ms`
- `ViClearExeMode = Disabled` - viClear() execution mode. Default: `execute_on_all`
- `OpcQueryAfterWrite = True` - same as driver.utilities.opc_query_after_write = True. Default: `False`
- `StbInErrorCheck = False` - if true, the driver checks errors with *STB? If false, it uses SYST:ERR?. Default: `True`

- `LoggingMode = On` - Sets the logging status right from the start. Default: `Off`
- `LoggingName = 'MyDevice'` - Sets the name to represent the session in the log entries. Default: `'resource_name'`
- `LogToGlobalTarget = True` - Sets the logging target to the class-property previously set with RsPulseSeq.set_global_logging_target() Default: `False`
- `LoggingToConsole = True` - Immediately starts logging to the console. Default: False
- `LoggingToUdp = True` - Immediately starts logging to the UDP port. Default: False
- `LoggingUdpPort = 49200` - UDP port to log to. Default: 49200

Parameters

- **resource_name** – VISA resource name, e.g. 'TCPIP::192.168.2.1::INSTR'
- **id_query** – if True, the instrument's model name is verified against the models supported by the driver and eventually throws an exception.
- **reset** – Resets the instrument (sends *RST command) and clears its status sybsystem.
- **options** – string tokens alternating the driver settings.
- **direct_session** – Another driver object or pyVisa object to reuse the session instead of opening a new session.

static assert_minimum_version(*min_version: str*) → None

Asserts that the driver version fulfills the minimum required version you have entered. This way you make sure your installed driver is of the entered version or newer.

classmethod clear_global_logging_relative_timestamp() → None

Clears the global relative timestamp. After this, all the instances using the global relative timestamp continue logging with the absolute timestamps.

close() → None

Closes the active RsPulseSeq session.

classmethod from_existing_session(*session: object*, *options: str = None*) → RsPulseSeq

Creates a new RsPulseSeq object with the entered 'session' reused.

Parameters

- **session** – can be another driver or a direct pyvisa session.
- **options** – string tokens alternating the driver settings.

classmethod get_global_logging_relative_timestamp() → datetime

Returns global common relative timestamp for log entries.

classmethod get_global_logging_target()

Returns global common target stream.

get_session_handle() → object

Returns the underlying session handle.

get_total_execution_time() → timedelta

Returns total time spent by the library on communicating with the instrument. This time is always shorter than get_total_time(), since it does not include gaps between the communication. You can reset this counter with reset_time_statistics().

get_total_time() → timedelta

Returns total time spent by the library on communicating with the instrument. This time is always shorter than get_total_time(), since it does not include gaps between the communication. You can reset this counter with reset_time_statistics().

static list_resources(*expression: str = '?*::INSTR'*, *visa_select: str = None*) → List[str]

Finds all the resources defined by the expression

- '?*' - matches all the available instruments
- 'USB::?*' - matches all the USB instruments
- 'TCPIP::192?*' - matches all the LAN instruments with the IP address starting with 192

Parameters

- **expression** – see the examples in the function
- **visa_select** – optional parameter selecting a specific VISA. Examples: '@ivi', '@rs'

reset_time_statistics() → None

Resets all execution and total time counters. Affects the results of get_total_time() and get_total_execution_time()

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

classmethod set_global_logging_relative_timestamp(*timestamp: datetime*) → None

Sets global common relative timestamp for log entries. To use it, call the following: io.utilities.logger.set_relative_timestamp_global()

classmethod set_global_logging_relative_timestamp_now() → None

Sets global common relative timestamp for log entries to this moment. To use it, call the following: io.utilities.logger.set_relative_timestamp_global().

classmethod set_global_logging_target(*target*) → None

Sets global common target stream that each instance can use. To use it, call the following: io.utilities.logger.set_logging_target_global(). If an instance uses global logging target, it automatically uses the global relative timestamp (if set). You can set the target to None to invalidate it.

Subgroups

5.1 Adjustment

class AdjustmentCls

Adjustment commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.adjustment.clone()
```

Subgroups

5.1.1 Reload

SCPI Command :

```
ADJustment:RELoad
```

class ReloadCls

Reload commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: ADJustment:RELoad
driver.adjustment.reload.set()
```

Reinitializes the adjustment database. You can create the level adjustment files not only automatically with the build-in Run Level Adjustment function but also externally (or manually) , by performing your own specific measurements. With this command, you can (re) load the level adjustment files, irrespectively of way they are created.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ADJustment:RELoad
driver.adjustment.reload.set_with_opc()
```

Reinitializes the adjustment database. You can create the level adjustment files not only automatically with the build-in Run Level Adjustment function but also externally (or manually) , by performing your own specific measurements. With this command, you can (re) load the level adjustment files, irrespectively of way they are created.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.2 Antenna

SCPI Commands :

```
ANTenna:CATalog
ANTenna:COMMent
ANTenna:CREate
ANTenna:NAME
```

(continues on next page)

(continued from previous page)

```
ANTenna:REMove
ANTenna:SELect
```

class AntennaCls

Antenna commands group definition. 87 total commands, 1 Subgroups, 6 group commands

get_catalog() → str

```
# SCPI: ANTenna:CATalog
value: str = driver.antenna.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: ANTenna:COMMent
value: str = driver.antenna.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_name() → str

```
# SCPI: ANTenna:NAME
value: str = driver.antenna.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: ANTenna:SELect
value: str = driver.antenna.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_comment(*comment: str*) → None

```
# SCPI: ANTenna:COMMent
driver.antenna.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: ANTenna:CREate
driver.antenna.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: ANTenna:NAME
driver.antenna.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: ANTenna:REMove
driver.antenna.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: ANTenna:SELect
driver.antenna.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.clone()
```

Subgroups

5.2.1 Model

SCPI Commands :

```
ANTenna:MODel:BANDwidth
ANTenna:MODel:FREQuency
ANTenna:MODel:POLarization
ANTenna:MODel:TYPE
```

class ModelCls

Model commands group definition. 81 total commands, 14 Subgroups, 4 group commands

get_bandwidth() → float

```
# SCPI: ANTenna:MODel:BANDwidth
value: float = driver.antenna.model.get_bandwidth()
```

Sets the antenna bandwidth.

return
bandwidth: float Range: 1e+06 to 1e+11

get_frequency() → float

```
# SCPI: ANTenna:MODel:FREQuency
value: float = driver.antenna.model.get_frequency()
```

Sets the frequency.

return
frequency: float Range: 1e+06 to 1e+11, Unit: Hz

get_polarization() → Polarization

```
# SCPI: ANTenna:MODel:POLarization
value: enums.Polarization = driver.antenna.model.get_polarization()
```

Sets the antenna polarization.

return
polarization: VERTical| HORizontal| CRIGht| CLEFt| SRIGht| SLEFt

get_type_py() → AntennaModel

```
# SCPI: ANTenna:MODel:TYPE
value: enums.AntennaModel = driver.antenna.model.get_type_py()
```

Sets the antenna pattern.

return
type_py: DIPole| PARabolic| GAUSsian| SINC| HORN| COSecant| ARRay| USER| CUSTom| CARRay| CARDoid| PLUGin

set_bandwidth(*bandwidth: float*) → None

```
# SCPI: ANTenna:MODel:BANDwidth
driver.antenna.model.set_bandwidth(bandwidth = 1.0)
```

Sets the antenna bandwidth.

param bandwidth
float Range: 1e+06 to 1e+11

set_frequency(*frequency: float*) → None

```
# SCPI: ANTenna:MODel:FREQuency
driver.antenna.model.set_frequency(frequency = 1.0)
```

Sets the frequency.

param frequency
float Range: 1e+06 to 1e+11, Unit: Hz

set_polarization(*polarization: Polarization*) → None

```
# SCPI: ANTenna:MODel:POLarization
driver.antenna.model.set_polarization(polarization = enums.Polarization.CLEFt)
```

Sets the antenna polarization.

param polarization
VERTical| HORizontal| CRIGht| CLEFt| SRIGht| SLEFt

set_type_py(*type_py: AntennaModel*) → None

```
# SCPI: ANTenna:MODel:TYPE
driver.antenna.model.set_type_py(type_py = enums.AntennaModel.ARRay)
```

Sets the antenna pattern.

param type_py
DIPole| PARabolic| GAUSsian| SINC| HORN| COSecant| ARRay| USER| CUSTom| CARRay| CARDoid| PLUGin

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.clone()
```

Subgroups

5.2.1.1 Array

SCPI Commands :

```
ANTenna:MODel:ARRay:NX
ANTenna:MODel:ARRay:NZ
ANTenna:MODel:ARRay:RESolution
ANTenna:MODel:ARRay:XDIStance
ANTenna:MODel:ARRay:ZDIStance
```

class ArrayCls

Array commands group definition. 16 total commands, 4 Subgroups, 5 group commands

get_nx() → float

```
# SCPI: ANTenna:MODel:ARRay:NX
value: float = driver.antenna.model.array.get_nx()
```

Sets the number of elements of the antenna array.

return
nx: No help available

get_nz() → float

```
# SCPI: ANTenna:MODel:ARRay:NZ
value: float = driver.antenna.model.array.get_nz()
```

Sets the number of elements of the antenna array.

return
nz: float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

get_resolution() → float

```
# SCPI: ANTenna:MODel:ARRay:RESolution
value: float = driver.antenna.model.array.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

get_xdistance() → float

```
# SCPI: ANTenna:MODel:ARRay:XDIStance
value: float = driver.antenna.model.array.get_xdistance()
```

Sets the spacing between the elements of the array antenna.

return
xdistance: No help available

get_zdistance() → float

```
# SCPI: ANTenna:MODel:ARRay:ZDIStance
value: float = driver.antenna.model.array.get_zdistance()
```

Sets the spacing between the elements of the array antenna.

return
zdistance: float Range: 0.0001 to 1

set_nx(*nx: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:NX
driver.antenna.model.array.set_nx(nx = 1.0)
```

Sets the number of elements of the antenna array.

param nx
float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

set_nz(*nz: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:NZ
driver.antenna.model.array.set_nz(nz = 1.0)
```

Sets the number of elements of the antenna array.

param nz
float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:RESolution
driver.antenna.model.array.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

set_xdistance(*xdistance: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:XDIStance
driver.antenna.model.array.set_xdistance(xdistance = 1.0)
```

Sets the spacing between the elements of the array antenna.

param xdistance
float Range: 0.0001 to 1

set_zdistance(*zdistance: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:ZDIStance
driver.antenna.model.array.set_zdistance(zdistance = 1.0)
```

Sets the spacing between the elements of the array antenna.

param zdistance
float Range: 0.0001 to 1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.array.clone()
```

Subgroups

5.2.1.1.1 Cosn

SCPI Commands :

```
ANTenna:MODel:ARRay:COSN:X
ANTenna:MODel:ARRay:COSN:Z
ANTenna:MODel:ARRay:COSN
```

class CosnCls

Cosn commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_value() → float

```
# SCPI: ANTenna:MODel:ARRay:COSN
value: float = driver.antenna.model.array.cosn.get_value()
```

Sets the value of the coefficient N in the cosN distribution.

return
cosn: float Range: 2 to 10

get_x() → float

```
# SCPI: ANTenna:MODel:ARRay:COSN:X
value: float = driver.antenna.model.array.cosn.get_x()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

return
x: float Range: 2 to 10

get_z() → float

```
# SCPI: ANTenna:MODel:ARRay:COSN:Z
value: float = driver.antenna.model.array.cosn.get_z()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

return
z: No help available

set_value(*cosn: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:COSN
driver.antenna.model.array.cosn.set_value(cosn = 1.0)
```

Sets the value of the coefficient N in the cosN distribution.

param cosn
float Range: 2 to 10

set_x(*x: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:COSN:X
driver.antenna.model.array.cosn.set_x(x = 1.0)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

param x
float Range: 2 to 10

set_z(*z: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:COSN:Z
driver.antenna.model.array.cosn.set_z(z = 1.0)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

param z
float Range: 2 to 10

5.2.1.1.2 Distribution

SCPI Commands :

```
ANTenna:MODel:ARRay:DISTribution:TYPE
ANTenna:MODel:ARRay:DISTribution:X
ANTenna:MODel:ARRay:DISTribution:Z
ANTenna:MODel:ARRay:DISTribution
```

class DistributionCls

Distribution commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_type_py() → bool

```
# SCPI: ANTenna:MODel:ARRay:DISTribution:TYPE
value: bool = driver.antenna.model.array.distribution.get_type_py()
```

Enables using the individual distribution function for X and Z direction.

return
type_py: ON| OFF| 1| 0

get_value() → AntennaModelArray

```
# SCPI: ANTenna:MODel:ARRay:DISTribution
value: enums.AntennaModelArray = driver.antenna.model.array.distribution.get_
↪value()
```

Sets the aperture distribution of the Planar Phased Array antenna.

return
distribution: UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

get_x() → AntennaModelArray

```
# SCPI: ANTenna:MODel:ARRay:DISTribution:X
value: enums.AntennaModelArray = driver.antenna.model.array.distribution.get_x()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

return
x: UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

get_z() → AntennaModelArray

```
# SCPI: ANTenna:MODel:ARRay:DISTribution:Z
value: enums.AntennaModelArray = driver.antenna.model.array.distribution.get_z()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

return
z: No help available

set_type_py(*type_py: bool*) → None

```
# SCPI: ANTenna:MODel:ARRay:DISTribution:TYPE
driver.antenna.model.array.distribution.set_type_py(type_py = False)
```

Enables using the individual distribution function for X and Z direction.

param type_py
ON| OFF| 1| 0

set_value(*distribution: AntennaModelArray*) → None

```
# SCPI: ANTenna:MODel:ARRay:DISTribution
driver.antenna.model.array.distribution.set_value(distribution = enums.
↪AntennaModelArray.COSine)
```

Sets the aperture distribution of the Planar Phased Array antenna.

param distribution
UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

set_x(*x: AntennaModelArray*) → None

```
# SCPI: ANTenna:MODel:ARRay:DISTribution:X
driver.antenna.model.array.distribution.set_x(x = enums.AntennaModelArray.
↪COSine)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

param x
UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

set_z(*z: AntennaModelArray*) → None

```
# SCPI: ANTenna:MODel:ARRay:DISTribution:Z
driver.antenna.model.array.distribution.set_z(z = enums.AntennaModelArray.
↪COSine)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

param z
UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

5.2.1.1.3 Element

SCPI Command :

```
ANTenna:MODel:ARRay:ELEMent:COSine
```

class ElementCls

Element commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_cosine() → bool

```
# SCPI: ANTenna:MODel:ARRay:ELEMent:COSine
value: bool = driver.antenna.model.array.element.get_cosine()
```

Sets the characteristic of individual antenna elements.

return
cosine: ON| OFF| 1| 0 0|OFF Omnidirectional characteristic 1|ON Cosine characteristic

set_cosine(*cosine: bool*) → None

```
# SCPI: ANTenna:MODel:ARRay:ELEMent:COSine
driver.antenna.model.array.element.set_cosine(cosine = False)
```

Sets the characteristic of individual antenna elements.

param cosine
ON| OFF| 1| 0 0|OFF Omnidirectional characteristic 1|ON Cosine characteristic

5.2.1.1.4 Pedestal

SCPI Commands :

```
ANTenna:MODel:ARRay:PEDestal:X
ANTenna:MODel:ARRay:PEDestal:Z
ANTenna:MODel:ARRay:PEDestal
```

class PedestalCls

Pedestal commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_value() → float

```
# SCPI: ANTenna:MODel:ARRay:PEDestal
value: float = driver.antenna.model.array.pedestal.get_value()
```

Sets the pedestal level of the antenna array.

return
pedestal: float Range: 0 to 1

get_x() → float

```
# SCPI: ANTenna:MODel:ARRay:PEDestal:X
value: float = driver.antenna.model.array.pedestal.get_x()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

return
x: float Range: 0 to 1

get_z() → float

```
# SCPI: ANTenna:MODel:ARRay:PEDestal:Z
value: float = driver.antenna.model.array.pedestal.get_z()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

return
z: No help available

set_value(*pedestal: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:PEDestal
driver.antenna.model.array.pedestal.set_value(pedestal = 1.0)
```

Sets the pedestal level of the antenna array.

param pedestal
float Range: 0 to 1

set_x(*x: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:PEDestal:X
driver.antenna.model.array.pedestal.set_x(x = 1.0)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

param x
float Range: 0 to 1

set_z(*z: float*) → None

```
# SCPI: ANTenna:MODel:ARRay:PEDestal:Z
driver.antenna.model.array.pedestal.set_z(z = 1.0)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

param z
float Range: 0 to 1

5.2.1.2 Backlobe

SCPI Commands :

```
ANTenna:MODel:BACKlobe:ATTenuation
ANTenna:MODel:BACKlobe:ENABle
ANTenna:MODel:BACKlobe:TYPE
```

class BacklobeCls

Backlobe commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_attenuation() → float

```
# SCPI: ANTenna:MODel:BACKlobe:ATTenuation
value: float = driver.antenna.model.backlobe.get_attenuation()
```

Sets the attenuation of the back lobe.

return
attenuation: float Range: 0 to 100

get_enable() → bool

```
# SCPI: ANTenna:MODel:BACKlobe:ENABle
value: bool = driver.antenna.model.backlobe.get_enable()
```

Enables the simulation of a back lobe.

return
enable: ON| OFF| 1| 0

get_type_py() → BlType

```
# SCPI: ANTenna:MODel:BACKlobe:TYPE
value: enums.BlType = driver.antenna.model.backlobe.get_type_py()
```

Sets the shape of the back lobe pattern.

return
type_py: MIRRor| OMNidirect

set_attenuation(*attenuation: float*) → None

```
# SCPI: ANTenna:MODel:BACKlobe:ATTenuation
driver.antenna.model.backlobe.set_attenuation(attenuation = 1.0)
```

Sets the attenuation of the back lobe.

param attenuation
float Range: 0 to 100

set_enable(*enable: bool*) → None

```
# SCPI: ANTenna:MODel:BACKlobe:ENABle
driver.antenna.model.backlobe.set_enable(enable = False)
```

Enables the simulation of a back lobe.

param enable
ON| OFF| 1| 0

set_type_py(*type_py: BlType*) → None

```
# SCPI: ANTenna:MODel:BACKlobe:TYPE
driver.antenna.model.backlobe.set_type_py(type_py = enums.BlType.MIRRor)
```

Sets the shape of the back lobe pattern.

param type_py
MIRRor| OMNidirect

5.2.1.3 Cardoid

SCPI Commands :

```
ANTenna:MODel:CARDoid:EXPonent
ANTenna:MODel:CARDoid:RESolution
```

class CardoidCls

Cardoid commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_exponent() → float

```
# SCPI: ANTenna:MODel:CARDoid:EXPonent
value: float = driver.antenna.model.cardoid.get_exponent()
```

Use values greater than 1 to narrow the antenna beam.

return
exponent: float Range: 1 to 20

get_resolution() → float

```
# SCPI: ANTenna:MODel:CARDoid:RESolution
value: float = driver.antenna.model.cardoid.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

set_exponent(*exponent: float*) → None

```
# SCPI: ANTenna:MODel:CARDoid:EXPonent
driver.antenna.model.cardoid.set_exponent(exponent = 1.0)
```

Use values greater than 1 to narrow the antenna beam.

param exponent
float Range: 1 to 20

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:CARDoid:RESolution
driver.antenna.model.cardoid.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

5.2.1.4 Carray

SCPI Commands :

```
ANTenna:MODel:CARRay:GEOMetry
ANTenna:MODel:CARRay:RESolution
```

class CarrayCls

Carray commands group definition. 25 total commands, 8 Subgroups, 2 group commands

get_geometry() → Geometry

```
# SCPI: ANTenna:MODel:CARRay:GEOMetry
value: enums.Geometry = driver.antenna.model.carray.get_geometry()
```

Sets the geometry of the custom phased array antenna.

return
geometry: RECTangular| LINear| HEXagonal| CIRCular

get_resolution() → float

```
# SCPI: ANTenna:MODel:CARRay:RESolution
value: float = driver.antenna.model.carray.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

set_geometry(*geometry: Geometry*) → None

```
# SCPI: ANTenna:MODel:CARRay:GEOMetry
driver.antenna.model.carray.set_geometry(geometry = enums.Geometry.CIRCular)
```

Sets the geometry of the custom phased array antenna.

param geometry
RECTangular| LINear| HEXagonal| CIRCular

`set_resolution`(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:RESolution
driver.antenna.model.carray.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.carray.clone()
```

Subgroups

5.2.1.4.1 Circular

SCPI Commands :

```
ANTenna:MODel:CARRay:CIRCular:DISTance
ANTenna:MODel:CARRay:CIRCular:LATTice
ANTenna:MODel:CARRay:CIRCular:RADius
```

`class CircularCls`

Circular commands group definition. 3 total commands, 0 Subgroups, 3 group commands

`get_distance`() → float

```
# SCPI: ANTenna:MODel:CARRay:CIRCular:DISTance
value: float = driver.antenna.model.carray.circular.get_distance()
```

Sets the spacing between the elements of the array antenna.

return
distance: No help available

`get_lattice`() → Lattice

```
# SCPI: ANTenna:MODel:CARRay:CIRCular:LATTice
value: enums.Lattice = driver.antenna.model.carray.circular.get_lattice()
```

Sets the lattice.

return
lattice: RECTangular| TRIangular

get_radius() → float

```
# SCPI: ANTenna:MODel:CARRay:CIRCular:RADius
value: float = driver.antenna.model.carray.circular.get_radius()
```

Set the radius of the circular phased array antenna.

return
radius: float Range: 1 to 50

set_distance(*distance: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:CIRCular:DISTance
driver.antenna.model.carray.circular.set_distance(distance = 1.0)
```

Sets the spacing between the elements of the array antenna.

param distance
float Range: 0.0001 to 1

set_lattice(*lattice: Lattice*) → None

```
# SCPI: ANTenna:MODel:CARRay:CIRCular:LATTice
driver.antenna.model.carray.circular.set_lattice(lattice = enums.Lattice.
↪RECTangular)
```

Sets the lattice.

param lattice
RECTangular| TRIangular

set_radius(*radius: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:CIRCular:RADius
driver.antenna.model.carray.circular.set_radius(radius = 1.0)
```

Set the radius of the circular phased array antenna.

param radius
float Range: 1 to 50

5.2.1.4.2 Cosn

SCPI Commands :

```
ANTenna:MODel:CARRay:COSN:X
ANTenna:MODel:CARRay:COSN:Z
ANTenna:MODel:CARRay:COSN
```

class CosnCls

Cosn commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_value() → float

```
# SCPI: ANTenna:MODel:CARRay:COSN
value: float = driver.antenna.model.carray.cosn.get_value()
```

Sets the value of the coefficient N in the cosN distribution.

return
cosn: float Range: 2 to 10

get_x() → float

```
# SCPI: ANTenna:MODel:CARRay:COSN:X
value: float = driver.antenna.model.carray.cosn.get_x()
```

Requires ANTenna:MODel:CARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

return
x: float Range: 2 to 10

get_z() → float

```
# SCPI: ANTenna:MODel:CARRay:COSN:Z
value: float = driver.antenna.model.carray.cosn.get_z()
```

Requires ANTenna:MODel:CARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

return
z: No help available

set_value(*cosn: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:COSN
driver.antenna.model.carray.cosn.set_value(cosn = 1.0)
```

Sets the value of the coefficient N in the cosN distribution.

param cosn
float Range: 2 to 10

set_x(*x: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:COSN:X
driver.antenna.model.carray.cosn.set_x(x = 1.0)
```

Requires ANTenna:MODel:CARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

param x
float Range: 2 to 10

set_z(*z: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:COSN:Z
driver.antenna.model.carray.cosn.set_z(z = 1.0)
```

Requires ANTenna:MODel:CARRay:DISTribution:TYPE 1. Sets the individual value of the coefficient N in the cosN distribution for X and Z direction.

param z
float Range: 2 to 10

5.2.1.4.3 Distribution

SCPI Commands :

```
ANTenna:MODel:CARRay:DISTribution:TYPE
ANTenna:MODel:CARRay:DISTribution:X
ANTenna:MODel:CARRay:DISTribution:Z
ANTenna:MODel:CARRay:DISTribution
```

class DistributionCls

Distribution commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_type_py() → bool

```
# SCPI: ANTenna:MODel:CARRay:DISTribution:TYPE
value: bool = driver.antenna.model.carray.distribution.get_type_py()
```

Enables using the individual distribution function for X and Z direction.

return
type_py: ON| OFF| 1| 0

get_value() → AntennaModelArray

```
# SCPI: ANTenna:MODel:CARRay:DISTribution
value: enums.AntennaModelArray = driver.antenna.model.carray.distribution.get_
↪value()
```

Sets the aperture distribution of the Custom Phased Array antenna.

return
distribution: UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

get_x() → AntennaModelArray

```
# SCPI: ANTenna:MODel:CARRay:DISTribution:X
value: enums.AntennaModelArray = driver.antenna.model.carray.distribution.get_
↪x()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

return
x: UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

get_z() → AntennaModelArray

```
# SCPI: ANTenna:MODel:CARRay:DISTribution:Z
value: enums.AntennaModelArray = driver.antenna.model.carray.distribution.get_
↪z()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

return
z: No help available

set_type_py(*type_py: bool*) → None

```
# SCPI: ANTenna:MODel:CARRay:DISTribution:TYPE
driver.antenna.model.carray.distribution.set_type_py(type_py = False)
```

Enables using the individual distribution function for X and Z direction.

param type_py
ON| OFF| 1| 0

set_value(*distribution: AntennaModelArray*) → None

```
# SCPI: ANTenna:MODel:CARRay:DISTribution
driver.antenna.model.carray.distribution.set_value(distribution = enums.
↪AntennaModelArray.COSine)
```

Sets the aperture distribution of the Custom Phased Array antenna.

param distribution
UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

set_x(*x: AntennaModelArray*) → None

```
# SCPI: ANTenna:MODel:CARRay:DISTribution:X
driver.antenna.model.carray.distribution.set_x(x = enums.AntennaModelArray.
↪COSine)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

param x
UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

set_z(*z: AntennaModelArray*) → None

```
# SCPI: ANTenna:MODel:CARRay:DISTribution:Z
driver.antenna.model.carray.distribution.set_z(z = enums.AntennaModelArray.
↪COSine)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual aperture distribution function for X and Z direction.

param z
UNIForm| PARabolic| COSine| CSQuared| COSN| TRIangular| HAMMing| HANN

5.2.1.4.4 Element

SCPI Command :

```
ANTenna:MODel:CARRay:ELEMent:COSine
```

class ElementCls

Element commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_cosine() → bool

```
# SCPI: ANTenna:MODel:CARRay:ELEMent:COSine
value: bool = driver.antenna.model.carray.element.get_cosine()
```

Sets the characteristic of individual antenna elements.

return
cosine: ON| OFF| 1| 0 0|OFF Omnidirectional characteristic 1|ON Cosine characteristic

set_cosine(*cosine: bool*) → None

```
# SCPI: ANTenna:MODel:CARRay:ELEMent:COSine
driver.antenna.model.carray.element.set_cosine(cosine = False)
```

Sets the characteristic of individual antenna elements.

param cosine
ON| OFF| 1| 0 0|OFF Omnidirectional characteristic 1|ON Cosine characteristic

5.2.1.4.5 Hexagonal

SCPI Commands :

```
ANTenna:MODel:CARRay:HEXagonal:DISTance
ANTenna:MODel:CARRay:HEXagonal:N
```

class HexagonalCls

Hexagonal commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_distance() → float

```
# SCPI: ANTenna:MODel:CARRay:HEXagonal:DISTance
value: float = driver.antenna.model.carray.hexagonal.get_distance()
```

Sets the spacing between the elements of the array antenna.

return
distance: No help available

get_n() → float

```
# SCPI: ANTenna:MODel:CARRay:HEXagonal:N
value: float = driver.antenna.model.carray.hexagonal.get_n()
```

Sets the number of elements of the antenna array.

return
n: No help available

set_distance(*distance: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:HEXagonal:DISTance
driver.antenna.model.carray.hexagonal.set_distance(distance = 1.0)
```

Sets the spacing between the elements of the array antenna.

param distance
float Range: 0.0001 to 1

set_n(*n: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:HEXagonal:N
driver.antenna.model.carray.hexagonal.set_n(n = 1.0)
```

Sets the number of elements of the antenna array.

param n
float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

5.2.1.4.6 Linear

SCPI Commands :

```
ANTenna:MODel:CARRay:LINear:DISTance
ANTenna:MODel:CARRay:LINear:N
```

class LinearCls

Linear commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_distance() → float

```
# SCPI: ANTenna:MODel:CARRay:LINear:DISTance
value: float = driver.antenna.model.carray.linear.get_distance()
```

Sets the spacing between the elements of the array antenna.

return
distance: No help available

get_n() → float

```
# SCPI: ANTenna:MODel:CARRay:LINear:N
value: float = driver.antenna.model.carray.linear.get_n()
```

Sets the number of elements of the antenna array.

return
n: No help available

set_distance(*distance: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:LINear:DISTance
driver.antenna.model.carray.linear.set_distance(distance = 1.0)
```

Sets the spacing between the elements of the array antenna.

param distance
float Range: 0.0001 to 1

set_n(*n: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:LINear:N
driver.antenna.model.carray.linear.set_n(n = 1.0)
```

Sets the number of elements of the antenna array.

param n
float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

5.2.1.4.7 Pedestal

SCPI Commands :

```
ANTenna:MODel:CARRay:PEDestal:X
ANTenna:MODel:CARRay:PEDestal:Z
ANTenna:MODel:CARRay:PEDestal
```

class PedestalCls

Pedestal commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_value() → float

```
# SCPI: ANTenna:MODel:CARRay:PEDestal
value: float = driver.antenna.model.carray.pedestal.get_value()
```

Sets the pedestal level of the antenna array.

return
pedestal: float Range: 0 to 1

get_x() → float

```
# SCPI: ANTenna:MODel:CARRay:PEDestal:X
value: float = driver.antenna.model.carray.pedestal.get_x()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

return
x: float Range: 0 to 1

get_z() → float

```
# SCPI: ANTenna:MODel:CARRay:PEDestal:Z
value: float = driver.antenna.model.carray.pedestal.get_z()
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

return
z: No help available

set_value(*pedestal: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:PEDestal
driver.antenna.model.carray.pedestal.set_value(pedestal = 1.0)
```

Sets the pedestal level of the antenna array.

param pedestal
float Range: 0 to 1

set_x(*x: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:PEDestal:X
driver.antenna.model.carray.pedestal.set_x(x = 1.0)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

param x
float Range: 0 to 1

set_z(*z: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:PEDestal:Z
driver.antenna.model.carray.pedestal.set_z(z = 1.0)
```

Requires ANTenna:MODel:ARRay:DISTribution:TYPE 1. Sets the individual pedestal level of the antenna array in X or Z direction.

param z
float Range: 0 to 1

5.2.1.4.8 Rectangular

SCPI Commands :

```
ANTenna:MODel:CARRay:RECTangular:LATTice
ANTenna:MODel:CARRay:RECTangular:NX
ANTenna:MODel:CARRay:RECTangular:NZ
ANTenna:MODel:CARRay:RECTangular:XDIStance
ANTenna:MODel:CARRay:RECTangular:ZDIStance
```

class RectangularCls

Rectangular commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_lattice() → Lattice

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:LATTice
value: enums.Lattice = driver.antenna.model.carray.rectangular.get_lattice()
```

Sets the lattice.

return
lattice: RECTangular| TRIangular

get_nx() → float

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:NX
value: float = driver.antenna.model.carray.rectangular.get_nx()
```

Sets the number of elements of the antenna array.

return
nx: No help available

get_nz() → float

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:NZ
value: float = driver.antenna.model.carray.rectangular.get_nz()
```

Sets the number of elements of the antenna array.

return
nz: float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

get_xdistance() → float

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:XDIStance
value: float = driver.antenna.model.carray.rectangular.get_xdistance()
```

Sets the spacing between the elements of the array antenna.

return
xdistance: No help available

get_zdistance() → float

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:ZDIStance
value: float = driver.antenna.model.carray.rectangular.get_zdistance()
```

Sets the spacing between the elements of the array antenna.

return
zdistance: float Range: 0.0001 to 1

set_lattice(*lattice: Lattice*) → None

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:LATTice
driver.antenna.model.carray.rectangular.set_lattice(lattice = enums.Lattice.
↪RECTangular)
```

Sets the lattice.

param lattice
RECTangular| TRIangular

set_nx(*nx: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:NX
driver.antenna.model.carray.rectangular.set_nx(nx = 1.0)
```

Sets the number of elements of the antenna array.

param nx
float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

set_nz(*nz: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:NZ
driver.antenna.model.carray.rectangular.set_nz(nz = 1.0)
```

Sets the number of elements of the antenna array.

param nz
float Range: 2 to 1000 (planar phased array; linear phase array) , 100 (rectangular phase array) , 50 (hexagonal phase array)

set_xdistance(*xdistance: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:XDIStance
driver.antenna.model.carray.rectangular.set_xdistance(xdistance = 1.0)
```

Sets the spacing between the elements of the array antenna.

param xdistance
float Range: 0.0001 to 1

set_zdistance(*zdistance: float*) → None

```
# SCPI: ANTenna:MODel:CARRay:RECTangular:ZDIStance
driver.antenna.model.carray.rectangular.set_zdistance(zdistance = 1.0)
```

Sets the spacing between the elements of the array antenna.

param zdistance
float Range: 0.0001 to 1

5.2.1.5 Cosecant

SCPI Commands :

```
ANTenna:MODel:COSecant:HPBW
ANTenna:MODel:COSecant:RESolution
ANTenna:MODel:COSecant:T1
ANTenna:MODel:COSecant:T2
```

class CosecantCls

Cosecant commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_hp_bw() → float

```
# SCPI: ANTenna:MODel:COSecant:HPBW
value: float = driver.antenna.model.cosecant.get_hp_bw()
```

Sets the Half-Power Beam Width Cosecant Squared antenna.

return
hp_bw: float Range: 0.01 to 30

get_resolution() → float

```
# SCPI: ANTenna:MODel:COSecant:RESolution
value: float = driver.antenna.model.cosecant.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

get_t_1() → float

```
# SCPI: ANTenna:MODel:COSecant:T1
value: float = driver.antenna.model.cosecant.get_t_1()
```

Sets the Theta parameters.

return
t_1: No help available

get_t_2() → float

```
# SCPI: ANTenna:MODel:COSecant:T2
value: float = driver.antenna.model.cosecant.get_t_2()
```

Sets the Theta parameters.

return
t_2: float Range: 1 to 90

set_hp_bw(*hp_bw: float*) → None

```
# SCPI: ANTenna:MODel:COSecant:HPBW
driver.antenna.model.cosecant.set_hp_bw(hp_bw = 1.0)
```

Sets the Half-Power Beam Width Cosecant Squared antenna.

param hp_bw
float Range: 0.01 to 30

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:COSecant:RESolution
driver.antenna.model.cosecant.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

set_t_1(*t_1: float*) → None

```
# SCPI: ANTenna:MODel:COSecant:T1
driver.antenna.model.cosecant.set_t_1(t_1 = 1.0)
```

Sets the Theta parameters.

param t_1
float Range: 1 to 90

set_t_2(*t_2: float*) → None

```
# SCPI: ANTenna:MODel:COSecant:T2
driver.antenna.model.cosecant.set_t_2(t_2 = 1.0)
```

Sets the Theta parameters.

param t_2
float Range: 1 to 90

5.2.1.6 Custom

SCPI Commands :

```
ANTenna:MODel:CUSTom:RESolution
ANTenna:MODel:CUSTom:SLRolloff
ANTenna:MODel:CUSTom:SLSCale
ANTenna:MODel:CUSTom:SLSTart
```

class CustomCls

Custom commands group definition. 6 total commands, 1 Subgroups, 4 group commands

get_resolution() → float

```
# SCPI: ANTenna:MODel:CUSTom:RESolution
value: float = driver.antenna.model.custom.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

get_sl_rolloff() → float

```
# SCPI: ANTenna:MODel:CUSTom:SLRolloff
value: float = driver.antenna.model.custom.get_sl_rolloff()
```

Sets the factor used to calculate the HPBW of the side lobes.

return
sl_rolloff: float Range: 1 to 45

get_sl_scale() → float

```
# SCPI: ANTenna:MODel:CUSTom:SLSCale
value: float = driver.antenna.model.custom.get_sl_scale()
```

Sets the step size to calculate the power level of the side lobes.

return
sl_scale: float Range: 0.01 to 10

get_sl_start() → float

```
# SCPI: ANTenna:MODel:CUSTom:SLSTart
value: float = driver.antenna.model.custom.get_sl_start()
```

Sets the power level of the first pairs of side lobes.

return
sl_start: float Range: 1 to 90

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:CUSTom:RESolution
driver.antenna.model.custom.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

set_sl_rolloff(*sl_rolloff: float*) → None

```
# SCPI: ANTenna:MODel:CUSTom:SLRolloff
driver.antenna.model.custom.set_sl_rolloff(sl_rolloff = 1.0)
```

Sets the factor used to calculate the HPBW of the side lobes.

param sl_rolloff
float Range: 1 to 45

set_sl_scale(*sl_scale: float*) → None

```
# SCPI: ANTenna:MODel:CUSTom:SLSCale
driver.antenna.model.custom.set_sl_scale(sl_scale = 1.0)
```

Sets the step size to calculate the power level of the side lobes.

param sl_scale
float Range: 0.01 to 10

set_sl_start(*sl_start: float*) → None

```
# SCPI: ANTenna:MODel:CUSTom:SLSTart
driver.antenna.model.custom.set_sl_start(sl_start = 1.0)
```

Sets the power level of the first pairs of side lobes.

param sl_start
float Range: 1 to 90

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.custom.clone()
```

Subgroups

5.2.1.6.1 HpBw

SCPI Commands :

```
ANTenna:MODel:CUSTom:HPBW:XY
ANTenna:MODel:CUSTom:HPBW:YZ
```

class HpBwCls

HpBw commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_xy() → float

```
# SCPI: ANTenna:MODel:CUSTom:HPBW:XY
value: float = driver.antenna.model.custom.hpBw.get_xy()
```

Sets the required HPBW of the custom antenna.

return
xy: No help available

get_yz() → float

```
# SCPI: ANTenna:MODel:CUSTom:HPBW:YZ
value: float = driver.antenna.model.custom.hpBw.get_yz()
```

Sets the required HPBW of the custom antenna.

return
yz: float Range: 0.1 to 45

set_xy(*xy: float*) → None

```
# SCPI: ANTenna:MODel:CUSTom:HPBW:XY
driver.antenna.model.custom.hpBw.set_xy(xy = 1.0)
```

Sets the required HPBW of the custom antenna.

param xy
float Range: 0.1 to 45

set_yz(*yz: float*) → None

```
# SCPI: ANTenna:MODel:CUSTom:HPBW:YZ
driver.antenna.model.custom.hpBw.set_yz(yz = 1.0)
```

Sets the required HPBW of the custom antenna.

param yz
float Range: 0.1 to 45

5.2.1.7 Dipole

SCPI Command :

```
ANTenna:MODel:DIPole:RESolution
```

class DipoleCls

Dipole commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_resolution() → float

```
# SCPI: ANTenna:MODel:DIPole:RESolution
value: float = driver.antenna.model.dipole.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:DIPole:RESolution
driver.antenna.model.dipole.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

5.2.1.8 Gaussian

SCPI Command :

```
ANTenna:MODel:GAUSsian:RESolution
```

class GaussianCls

Gaussian commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_resolution() → float

```
# SCPI: ANTenna:MODel:GAUSsian:RESolution
value: float = driver.antenna.model.gaussian.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:GAUSsian:RESolution
driver.antenna.model.gaussian.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.gaussian.clone()
```

Subgroups

5.2.1.8.1 HpBw

SCPI Commands :

```
ANTenna:MODel:GAUSsian:HPBW:AZIMuth
ANTenna:MODel:GAUSsian:HPBW:ELEVation
```

class HpBwCls

HpBw commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_azimuth() → float

```
# SCPI: ANTenna:MODel:GAUSsian:HPBW:AZIMuth
value: float = driver.antenna.model.gaussian.hpBw.get_azimuth()
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

return
azimuth: No help available

get_elevation() → float

```
# SCPI: ANTenna:MODel:GAUSsian:HPBW:ELEVation
value: float = driver.antenna.model.gaussian.hpBw.get_elevation()
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

return
elevation: float Range: 0.1 to 45, Unit: degree

set_azimuth(*azimuth: float*) → None

```
# SCPI: ANTenna:MODel:GAUSsian:HPBW:AZIMuth
driver.antenna.model.gaussian.hpBw.set_azimuth(azimuth = 1.0)
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

param azimuth
float Range: 0.1 to 45, Unit: degree

set_elevation(*elevation: float*) → None

```
# SCPI: ANTenna:MODel:GAUSsian:HPBW:ELEVation
driver.antenna.model.gaussian.hpBw.set_elevation(elevation = 1.0)
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

param elevation
float Range: 0.1 to 45, Unit: degree

5.2.1.9 Horn

SCPI Commands :

```
ANTenna:MODel:HORN:LX
ANTenna:MODel:HORN:LZ
ANTenna:MODel:HORN:RESolution
```

class HornCls

Horn commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_lx() → float

```
# SCPI: ANTenna:MODel:HORN:LX
value: float = driver.antenna.model.horn.get_lx()
```

Sets the length of the rectangular sides of the Pyramidal Horn antenna.

return
lx: No help available

get_lz() → float

```
# SCPI: ANTenna:MODel:HORN:LZ
value: float = driver.antenna.model.horn.get_lz()
```

Sets the length of the rectangular sides of the Pyramidal Horn antenna.

return
lz: float Range: 0.01 to 100, Unit: m

get_resolution() → float

```
# SCPI: ANTenna:MODel:HORN:RESolution
value: float = driver.antenna.model.horn.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

set_lx(*lx: float*) → None

```
# SCPI: ANTenna:MODel:HORN:LX
driver.antenna.model.horn.set_lx(lx = 1.0)
```

Sets the length of the rectangular sides of the Pyramidal Horn antenna.

param lx
float Range: 0.01 to 100, Unit: m

set_lz(*lz: float*) → None

```
# SCPI: ANTenna:MODel:HORN:LZ
driver.antenna.model.horn.set_lz(lz = 1.0)
```

Sets the length of the rectangular sides of the Pyramidal Horn antenna.

param lz
float Range: 0.01 to 100, Unit: m

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:HORN:RESolution
driver.antenna.model.horn.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

5.2.1.10 Parabolic

SCPI Commands :

```
ANTenna:MODel:PARabolic:DIAMeter
ANTenna:MODel:PARabolic:RESolution
```

class ParabolicCls

Parabolic commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_diameter() → float

```
# SCPI: ANTenna:MODel:PARabolic:DIAMeter
value: float = driver.antenna.model.parabolic.get_diameter()
```

Sets the diameter of the parabolic dish antenna.

return
diameter: float Range: 0.05 to 100, Unit: m

get_resolution() → float

```
# SCPI: ANTenna:MODel:PARabolic:RESolution
value: float = driver.antenna.model.parabolic.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

set_diameter(*diameter: float*) → None

```
# SCPI: ANTenna:MODel:PARabolic:DIAMeter
driver.antenna.model.parabolic.set_diameter(diameter = 1.0)
```

Sets the diameter of the parabolic dish antenna.

param diameter
float Range: 0.05 to 100, Unit: m

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:PARabolic:RESolution
driver.antenna.model.parabolic.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

5.2.1.11 Plugin

SCPI Command :

```
ANTenna:MODel:PLUGin:NAME
```

class PluginCls

Plugin commands group definition. 4 total commands, 1 Subgroups, 1 group commands

get_name() → str

```
# SCPI: ANTenna:MODel:PLUGin:NAME
value: str = driver.antenna.model.plugin.get_name()
```

No command help available

return
name: No help available

set_name(*name: str*) → None

```
# SCPI: ANTenna:MODel:PLUGin:NAME
driver.antenna.model.plugin.set_name(name = 'abc')
```

No command help available

param name
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.plugin.clone()
```

Subgroups

5.2.1.11.1 Variable

SCPI Commands :

```
ANTenna:MODel:PLUGin:VARiable:CATalog
ANTenna:MODel:PLUGin:VARiable:SELect
ANTenna:MODel:PLUGin:VARiable:VALue
```

class VariableCls

Variable commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_catalog() → str

```
# SCPI: ANTenna:MODel:PLUGin:VARiable:CATalog
value: str = driver.antenna.model.plugin.variable.get_catalog()
```

Queries the variables used in the plugin.

return
catalog: string

get_select() → str

```
# SCPI: ANTenna:MODel:PLUGin:VARiable:SELect
value: str = driver.antenna.model.plugin.variable.get_select()
```

No command help available

return
select: No help available

get_value() → str

```
# SCPI: ANTenna:MODel:PLUGin:VARiable:VALue
value: str = driver.antenna.model.plugin.variable.get_value()
```

No command help available

return
value: No help available

set_select(*select: str*) → None

```
# SCPI: ANTenna:MODel:PLUGin:VARiable:SELect
driver.antenna.model.plugin.variable.set_select(select = 'abc')
```

No command help available

param select
No help available

set_value(*value: str*) → None

```
# SCPI: ANTenna:MODel:PLUGin:VARiable:VALue
driver.antenna.model.plugin.variable.set_value(value = 'abc')
```

No command help available

param value
No help available

5.2.1.12 Rotation

SCPI Commands :

```
ANTenna:MODel:ROTation:X
ANTenna:MODel:ROTation:Z
```

class RotationCls

Rotation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_x() → float

```
# SCPI: ANTenna:MODel:ROTation:X
value: float = driver.antenna.model.rotation.get_x()
```

Sets the X and Z antenna rotation.

return
x: No help available

get_z() → float

```
# SCPI: ANTenna:MODel:ROTation:Z
value: float = driver.antenna.model.rotation.get_z()
```

Sets the X and Z antenna rotation.

return
z: float Range: -180 to 180, Unit: degree

set_x(*x: float*) → None

```
# SCPI: ANTenna:MODel:ROTation:X
driver.antenna.model.rotation.set_x(x = 1.0)
```

Sets the X and Z antenna rotation.

param x
float Range: -180 to 180, Unit: degree

set_z(*z: float*) → None

```
# SCPI: ANTenna:MODel:ROTation:Z
driver.antenna.model.rotation.set_z(z = 1.0)
```

Sets the X and Z antenna rotation.

param z
float Range: -180 to 180, Unit: degree

5.2.1.13 Sinc

SCPI Command :

```
ANTenna:MODel:SINC:RESolution
```

class SincCls

Sinc commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_resolution() → float

```
# SCPI: ANTenna:MODel:SINC:RESolution
value: float = driver.antenna.model.sinc.get_resolution()
```

Sets a custom resolution for the antenna pattern simulation.

return
resolution: float Range: 0.1 to 1

set_resolution(*resolution: float*) → None

```
# SCPI: ANTenna:MODel:SINC:RESolution
driver.antenna.model.sinc.set_resolution(resolution = 1.0)
```

Sets a custom resolution for the antenna pattern simulation.

param resolution
float Range: 0.1 to 1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.sinc.clone()
```

Subgroups

5.2.1.13.1 HpBw

SCPI Commands :

```
ANTenna:MODel:SINC:HPBW:AZIMuth
ANTenna:MODel:SINC:HPBW:ELEVation
```

class HpBwCls

HpBw commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_azimuth() → float

```
# SCPI: ANTenna:MODel:SINC:HPBW:AZIMuth
value: float = driver.antenna.model.sinc.hpBw.get_azimuth()
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

return
azimuth: No help available

`get_elevation`() → float

```
# SCPI: ANTenna:MODel:SINC:HPBW:ELEVation
value: float = driver.antenna.model.sinc.hpBw.get_elevation()
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

return
elevation: float Range: 0.1 to 45, Unit: degree

`set_azimuth`(*azimuth: float*) → None

```
# SCPI: ANTenna:MODel:SINC:HPBW:AZIMuth
driver.antenna.model.sinc.hpBw.set_azimuth(azimuth = 1.0)
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

param azimuth
float Range: 0.1 to 45, Unit: degree

`set_elevation`(*elevation: float*) → None

```
# SCPI: ANTenna:MODel:SINC:HPBW:ELEVation
driver.antenna.model.sinc.hpBw.set_elevation(elevation = 1.0)
```

Sets the Half-Power Beam Width in azimuth and elevation direction for the Gaussian and Sin(x) /x antennas.

param elevation
float Range: 0.1 to 45, Unit: degree

5.2.1.14 User

SCPI Commands :

```
ANTenna:MODel:USER:CLEar
ANTenna:MODel:USER:LOAD
```

`class UserCls`

User commands group definition. 3 total commands, 1 Subgroups, 2 group commands

`clear`() → None

```
# SCPI: ANTenna:MODel:USER:CLEar
driver.antenna.model.user.clear()
```

Deletes all items from the list or the table.

`clear_with_opc`(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ANTenna:MODel:USER:CLEar
driver.antenna.model.user.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

load(*load: str*) → None

```
# SCPI: ANTenna:MODel:USER:LOAD
driver.antenna.model.user.load(load = 'abc')
```

Loads a custom antenna pattern file.

param load
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.antenna.model.user.clone()
```

Subgroups

5.2.1.14.1 Csv

SCPI Command :

```
ANTenna:MODel:USER:CSV:FORMat
```

class CsvCls

Csv commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set_format_py(*format_py: List[str]*) → None

```
# SCPI: ANTenna:MODel:USER:CSV:FORMat
driver.antenna.model.user.csv.set_format_py(format_py = ['abc1', 'abc2', 'abc3
↪'])
```

Defines how the data in the selected *.csv file is interpreted. The settings in this command are not permanent. The command affects only the currently selected antenna. For description of the *.csv file format, see 'Antenna pattern file formats'.

param format_py
No help available

5.3 ArbComposer

class ArbComposerCls

ArbComposer commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.arbComposer.clone()
```

Subgroups

5.3.1 Show

SCPI Command :

```
ARBComposer:SHOW
```

class ShowCls

Show commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: ARBComposer:SHOW
driver.arbComposer.show.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ARBComposer:SHOW
driver.arbComposer.show.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.4 Assignment

class AssignmentCls

Assignment commands group definition. 35 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.clone()
```

Subgroups

5.4.1 Antennas

SCPI Commands :

```
ASSignment:ANTennas:LIST
ASSignment:ANTennas:SELect
```

class AntennasCls

Antennas commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_list_py() → str

```
# SCPI: ASSignment:ANTennas:LIST
value: str = driver.assignment.antennas.get_list_py()
```

Queries the alias names of the unassigned receiver signals.

return
list_py: 'ReceiverSignal#1','ReceiverSignal#2',...

get_select() → str

```
# SCPI: ASSignment:ANTennas:SELect
value: str = driver.assignment.antennas.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:ANTennas:SELect
driver.assignment.antennas.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

5.4.2 Destination

SCPI Commands :

```
ASSignment:DESTination:LIST
ASSignment:DESTination:SELect
```

class DestinationCls

Destination commands group definition. 14 total commands, 1 Subgroups, 2 group commands

get_list_py() → str

```
# SCPI: ASSignment:DESTination:LIST
value: str = driver.assignment.destination.get_list_py()
```

Queries a list of the available destinations.

return
list_py: 'GenName#1','GenName2',...

get_select() → str

```
# SCPI: ASSignment:DESTination:SELect
value: str = driver.assignment.destination.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:DESTination:SELect
driver.assignment.destination.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.destination.clone()
```

Subgroups

5.4.2.1 Path

SCPI Commands :

```
ASSignment:DESTination:PATH:LIST
ASSignment:DESTination:PATH:SELect
```

class PathCls

Path commands group definition. 12 total commands, 2 Subgroups, 2 group commands

get_list_py() → str

```
# SCPI: ASSignment:DESTination:PATH:LIST
value: str = driver.assignment.destination.path.get_list_py()
```

Queries the available paths.

return
list_py: 'Path#1','Path#2',... List of available paths.

get_select() → str

```
# SCPI: ASSignment:DESTination:PATH:SELect
value: str = driver.assignment.destination.path.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:DESTination:PATH:SELect
driver.assignment.destination.path.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.destination.path.clone()
```

Subgroups

5.4.2.1.1 Antenna

SCPI Commands :

```
ASSignment:DESTination:PATH:ANTenna:CLEar
ASSignment:DESTination:PATH:ANTenna:DELete
ASSignment:DESTination:PATH:ANTenna:LIST
ASSignment:DESTination:PATH:ANTenna:SELect
```

class AntennaCls

Antenna commands group definition. 5 total commands, 1 Subgroups, 4 group commands

clear() → None

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:CLEar
driver.assignment.destination.path.antenna.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:CLEar
driver.assignment.destination.path.antenna.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete() → None

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:DELete
driver.assignment.destination.path.antenna.delete()
```

Deletes the particular item.

delete_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:DELete
driver.assignment.destination.path.antenna.delete_with_opc()
```

Deletes the particular item.

Same as delete, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_list_py() → str

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:LIST
value: str = driver.assignment.destination.path.antenna.get_list_py()
```

Queries the list of assigned receiver signals to the selected plugin.

return
list_py: 'ReceiverSignal#1','ReceiverSignal#2',...

get_select() → str

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:SELect
value: str = driver.assignment.destination.path.antenna.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:SELect
driver.assignment.destination.path.antenna.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.destination.path.antenna.clone()
```

Subgroups

5.4.2.1.1.1 Add

SCPI Command :

```
ASSignment:DESTination:PATH:ANTenna:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:ADD
driver.assignment.destination.path.antenna.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:DESTination:PATH:ANTenna:ADD
driver.assignment.destination.path.antenna.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.4.2.1.2 Emitter

SCPI Commands :

```
ASSignment:DESTination:PATH:EMITter:CLEar
ASSignment:DESTination:PATH:EMITter:DELete
ASSignment:DESTination:PATH:EMITter:LIST
ASSignment:DESTination:PATH:EMITter:SELect
```

class EmitterCls

Emitter commands group definition. 5 total commands, 1 Subgroups, 4 group commands

clear() → None

```
# SCPI: ASSignment:DESTination:PATH:EMITter:CLEar
driver.assignment.destination.path.emitter.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:DESTination:PATH:EMITter:CLEar
driver.assignment.destination.path.emitter.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete() → None

```
# SCPI: ASSignment:DESTination:PATH:EMITter:DELete
driver.assignment.destination.path.emitter.delete()
```

Deletes the particular item.

delete_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:DESTination:PATH:EMITter:DELete
driver.assignment.destination.path.emitter.delete_with_opc()
```

Deletes the particular item.

Same as delete, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_list_py() → str

```
# SCPI: ASSignment:DESTination:PATH:EMITter:LIST
value: str = driver.assignment.destination.path.emitter.get_list_py()
```

Queries the list of assigned emitters to the selected path.

return
list_py: 'Emitter/Inter#1','Emitter/Inter#2',...

get_select() → str

```
# SCPI: ASSignment:DESTination:PATH:EMITter:SELect
value: str = driver.assignment.destination.path.emitter.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:DESTination:PATH:EMITter:SELect
driver.assignment.destination.path.emitter.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Destination.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.destination.path.emitter.clone()
```

Subgroups

5.4.2.1.2.1 Add

SCPI Command :

```
ASSignment:DESTination:PATH:EMITter:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: ASSignment:DESTination:PATH:EMITter:ADD
driver.assignment.destination.path.emitter.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:DESTination:PATH:EMITter:ADD
driver.assignment.destination.path.emitter.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.4.3 Emitters

SCPI Commands :

```
ASSignment:EMITters:LIST
ASSignment:EMITters:SELect
```

class EmittersCls

Emitters commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_list_py() → str

```
# SCPI: ASSignment:EMITters:LIST
value: str = driver.assignment.emitters.get_list_py()
```

Queries the alias names of the unassigned emitters/interferers.

return
list_py: 'Emitter/Inter#1','Emitter/Inter#2',...

get_select() → str

```
# SCPI: ASSignment:EMITters:SELect
value: str = driver.assignment.emitters.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command.
For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:EMITters:SELect
driver.assignment.emitters.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

5.4.4 Generator

SCPI Commands :

```
ASSignment:GENerator:CAPabilities
ASSignment:GENerator:LIST
ASSignment:GENerator:SELect
```

class GeneratorCls

Generator commands group definition. 15 total commands, 1 Subgroups, 3 group commands

get_capabilities() → str

```
# SCPI: ASSignment:GENerator:CAPabilities
value: str = driver.assignment.generator.get_capabilities()
```

Queries the capabilities of the selected generator.

return
capabilities: RFA, BWA, MEMA, SWA, [RFB, BWB, MEMB, SWB] String that lists the maximum RF and BW, available memory size, and installed options per path.

get_list_py() → str

```
# SCPI: ASSignment:GENerator:LIST
value: str = driver.assignment.generator.get_list_py()
```

Queries a list of the available generators.

return
list_py: 'GenName#1','GenName2',...

get_select() → str

```
# SCPI: ASSignment:GENerator:SELect
value: str = driver.assignment.generator.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:GENerator:SELect
driver.assignment.generator.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.generator.clone()
```

Subgroups

5.4.4.1 Path

SCPI Commands :

```
ASSignment:GENerator:PATH:LIST
ASSignment:GENerator:PATH:SELect
```

class PathCls

Path commands group definition. 12 total commands, 2 Subgroups, 2 group commands

get_list_py() → str

```
# SCPI: ASSignment:GENerator:PATH:LIST
value: str = driver.assignment.generator.path.get_list_py()
```

Queries the available paths.

return
list_py: 'Path#1','Path#2',... List of available paths.

get_select() → str

```
# SCPI: ASSignment:GENerator:PATH:SELect
value: str = driver.assignment.generator.path.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:GENerator:PATH:SELect
driver.assignment.generator.path.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.generator.path.clone()
```

Subgroups

5.4.4.1.1 Antenna

SCPI Commands :

```
ASSignment:GENerator:PATH:ANTenna:CLEar
ASSignment:GENerator:PATH:ANTenna:DELete
ASSignment:GENerator:PATH:ANTenna:LIST
ASSignment:GENerator:PATH:ANTenna:SELect
```

class AntennaCls

Antenna commands group definition. 5 total commands, 1 Subgroups, 4 group commands

clear() → None

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:CLEar
driver.assignment.generator.path.antenna.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:CLEar
driver.assignment.generator.path.antenna.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete() → None

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:DELete
driver.assignment.generator.path.antenna.delete()
```

Deletes the particular item.

delete_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:DELete
driver.assignment.generator.path.antenna.delete_with_opc()
```

Deletes the particular item.

Same as delete, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_list_py() → str

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:LIST
value: str = driver.assignment.generator.path.antenna.get_list_py()
```

Queries the list of assigned receiver signals to the selected path.

return
list_py: 'ReceiverSignal#1','ReceiverSignal#2',...

get_select() → str

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:SELect
value: str = driver.assignment.generator.path.antenna.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:SELect
driver.assignment.generator.path.antenna.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.generator.path.antenna.clone()
```

Subgroups

5.4.4.1.1.1 Add

SCPI Command :

```
ASSignment:GENerator:PATH:ANTenna:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:ADD
driver.assignment.generator.path.antenna.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:GENerator:PATH:ANTenna:ADD
driver.assignment.generator.path.antenna.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.4.4.1.2 Emitter

SCPI Commands :

```
ASSignment:GENerator:PATH:EMITter:CLEar
ASSignment:GENerator:PATH:EMITter:DELete
ASSignment:GENerator:PATH:EMITter:LIST
ASSignment:GENerator:PATH:EMITter:SELect
```

class EmitterCls

Emitter commands group definition. 5 total commands, 1 Subgroups, 4 group commands

clear() → None

```
# SCPI: ASSignment:GENerator:PATH:EMITter:CLEar
driver.assignment.generator.path.emitter.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:GENerator:PATH:EMITter:CLEar
driver.assignment.generator.path.emitter.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete() → None

```
# SCPI: ASSignment:GENerator:PATH:EMITter:DELete
driver.assignment.generator.path.emitter.delete()
```

Deletes the particular item.

delete_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:GENerator:PATH:EMITter:DELete
driver.assignment.generator.path.emitter.delete_with_opc()
```

Deletes the particular item.

Same as delete, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_list_py() → str

```
# SCPI: ASSignment:GENerator:PATH:EMITter:LIST
value: str = driver.assignment.generator.path.emitter.get_list_py()
```

Queries the list of assigned emitters/interferes to the selected path.

return
list_py: 'Emitter/Inter#1','Emitter/Inter#2',...

get_select() → str

```
# SCPI: ASSignment:GENerator:PATH:EMITter:SELect
value: str = driver.assignment.generator.path.emitter.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: ASSignment:GENerator:PATH:EMITter:SELect
driver.assignment.generator.path.emitter.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.assignment.generator.path.emitter.clone()
```

Subgroups

5.4.4.1.2.1 Add

SCPI Command :

```
ASSignment:GENerator:PATH:EMITter:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: ASSignment:GENerator:PATH:EMITter:ADD
driver.assignment.generator.path.emitter.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ASSignment:GENerator:PATH:EMITter:ADD
driver.assignment.generator.path.emitter.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.4.5 Group

SCPI Commands :

```
ASSignment:GROup:LIST
ASSignment:GROup:SELect
```

class GroupCls

Group commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_list_py() → str

```
# SCPI: ASSignment:GROup:LIST
value: str = driver.assignment.group.get_list_py()
```

If interleaving groups are defined, queries the alias names of the unassigned interleaving groups.

return
list_py: string

get_select() → str

```
# SCPI: ASSignment:GROup:SELect
value: str = driver.assignment.group.get_select()
```

Assigns the selected group to the plugin and path selected with the commands method RsPulseSeq.Assignment.Destination. select and method RsPulseSeq.Assignment.Destination.Path.select.

return
select: string

set_select(*select: str*) → None

```
# SCPI: ASSignment:GROup:SELect
driver.assignment.group.set_select(select = 'abc')
```

Assigns the selected group to the plugin and path selected with the commands method RsPulseSeq.Assignment.Destination. select and method RsPulseSeq.Assignment.Destination.Path.select.

param select
string

5.5 Cpanel

SCPI Commands :

```
CPANel:ACTivate
CPANel:DEACtivate
CPANel:SETup
```

class CpanelCls

Cpanel commands group definition. 25 total commands, 6 Subgroups, 3 group commands

activate() → None

```
# SCPI: CPANel:ACTivate
driver.cpanel.activate()
```

Activates and deactivates the remote control of the control panel. Further CPANel:... commands cannot be executed, if the remote control is not active. After configuration, always deactivate the remote control of the control panel.

activate_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CPANel:ACTivate
driver.cpanel.activate_with_opc()
```

Activates and deactivates the remote control of the control panel. Further CPANel:... commands cannot be executed, if the remote control is not active. After configuration, always deactivate the remote control of the control panel.

Same as activate, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

deactivate() → None

```
# SCPI: CPANel:DEACtivate
driver.cpanel.deactivate()
```

Activates and deactivates the remote control of the control panel. Further CPANel:... commands cannot be executed, if the remote control is not active. After configuration, always deactivate the remote control of the control panel.

deactivate_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CPANel:DEACtivate
driver.cpanel.deactivate_with_opc()
```

Activates and deactivates the remote control of the control panel. Further CPANel:... commands cannot be executed, if the remote control is not active. After configuration, always deactivate the remote control of the control panel.

Same as deactivate, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_setup() → str

```
# SCPI: CPANel:SETup
value: str = driver.cpanel.get_setup()
```

Queries the name of the setup selected in the 'Signal Generators' dialog.

return
setup: string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.cpanel.clone()
```

Subgroups

5.5.1 Mute

SCPI Command :

```
CPANel:MUTE
```

class MuteCls

Mute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CPANel:MUTE
driver.cpanel.mute.set()
```

Deactivates the RF outputs of all signal generators.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CPANel:MUTE
driver.cpanel.mute.set_with_opc()
```

Deactivates the RF outputs of all signal generators.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.5.2 Refresh

SCPI Command :

```
CPANel:REFResh
```

class RefreshCls

Refresh commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CPANel:REFResh
driver.cpanel.refresh.set()
```

Refreshes the displayed information.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CPANel:REFResh
driver.cpanel.refresh.set_with_opc()
```

Refreshes the displayed information.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.5.3 Scenario

class ScenarioCls

Scenario commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.cpanel.scenario.clone()
```

Subgroups

5.5.3.1 Current

SCPI Commands :

```
CPANel:SCENario:CURRent:NAME
CPANel:SCENario:CURRent:STATus
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_name() → str

```
# SCPI: CPANel:SCENario:CURRent:NAME
value: str = driver.cpanel.scenario.current.get_name()
```

Queries the name of the current/last deployed scenario.

return
name: string

get_status() → str

```
# SCPI: CPANel:SCENario:CURRent:STATus
value: str = driver.cpanel.scenario.current.get_status()
```

Queries the scenario status.

return
status: string

5.5.3.2 Deployed

SCPI Commands :

```
CPANel:SCENario:DEPLoyed:NAME
CPANel:SCENario:DEPLoyed:TIME
```

class DeployedCls

Deployed commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_name() → str

```
# SCPI: CPANel:SCENario:DEPLoyed:NAME
value: str = driver.cpanel.scenario.deployed.get_name()
```

Queries the name of the current/last deployed scenario.

return
name: string

get_time() → str

```
# SCPI: CPANel:SCENario:DEPLoyed:TIME
value: str = driver.cpanel.scenario.deployed.get_time()
```

Queries the date and time the scenario has been deployed.

return
time: DD MMM YYY hh:mm:ss

5.5.4 Unmute

SCPI Command :

```
CPANel:UNMute
```

class UnmuteCls

Unmute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CPANel:UNMute
driver.cpanel.unmute.set()
```

Activates the RF outputs of all signal generators.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CPANel:UNMute
driver.cpanel.unmute.set_with_opc()
```

Activates the RF outputs of all signal generators.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.5.5 Unused

SCPI Commands :

```
CPANel:UNUSed:ALL
CPANel:UNUSed:FREQuency
CPANel:UNUSed:LEVel
CPANel:UNUSed:LIST
CPANel:UNUSed:SELect
CPANel:UNUSed:STATe
```

class UnusedCls

Unused commands group definition. 8 total commands, 1 Subgroups, 6 group commands

get_all() → bool

```
# SCPI: CPANel:UNUSed:ALL
value: bool = driver.cpanel.unused.get_all()
```

Shows all unused signal generators.

return
all_py: ON| OFF| 1| 0

get_frequency() → float

```
# SCPI: CPANel:UNUSed:FREQuency
value: float = driver.cpanel.unused.get_frequency()
```

Sets the RF frequency.

return
frequency: float

get_level() → float

```
# SCPI: CPANel:UNUSed:LEVel
value: float = driver.cpanel.unused.get_level()
```

Sets the RF level.

return
level: float

get_list_py() → List[str]

```
# SCPI: CPANel:UNUSed:LIST
value: List[str] = driver.cpanel.unused.get_list_py()
```

Queries the names of the used/unused signal generators.

return
list_py: 'Instr#1','Instr#2',...

get_select() → str

```
# SCPI: CPANel:UNUSed:SELect
value: str = driver.cpanel.unused.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

get_state() → bool

```
# SCPI: CPANel:UNUSed:STATe
value: bool = driver.cpanel.unused.get_state()
```

Activates the RF output of the selected signal generator.

return
state: ON| OFF| 1| 0

set_all(*all_py: bool*) → None

```
# SCPI: CPANel:UNUSed:ALL
driver.cpanel.unused.set_all(all_py = False)
```

Shows all unused signal generators.

param all_py
ON| OFF| 1| 0

set_frequency(*frequency: float*) → None

```
# SCPI: CPANel:UNUSed:FREQuency
driver.cpanel.unused.set_frequency(frequency = 1.0)
```

Sets the RF frequency.

param frequency
float

set_level(*level: float*) → None

```
# SCPI: CPANel:UNUSed:LEVel
driver.cpanel.unused.set_level(level = 1.0)
```

Sets the RF level.

param level
float

set_select(*select: str*) → None

```
# SCPI: CPANel:UNUSed:SELect
driver.cpanel.unused.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_state(*state: bool*) → None

```
# SCPI: CPANel:UNUSed:STATe
driver.cpanel.unused.set_state(state = False)
```

Activates the RF output of the selected signal generator.

param state
ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.cpanel.unused.clone()
```

Subgroups

5.5.5.1 Path

SCPI Commands :

```
CPANel:UNUSed:PATH:LIST
CPANel:UNUSed:PATH:SELect
```

class PathCls

Path commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_list_py() → str

```
# SCPI: CPANel:UNUSed:PATH:LIST
value: str = driver.cpanel.unused.path.get_list_py()
```

Queries the available RF paths of the selected signal generators.

return
list_py: 'Path#1','Paht#2',...

get_select() → str

```
# SCPI: CPANel:UNUSed:PATH:SELect
value: str = driver.cpanel.unused.path.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command.
For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_list_py(*list_py: str*) → None

```
# SCPI: CPANel:UNUSed:PATH:LIST
driver.cpanel.unused.path.set_list_py(list_py = 'abc')
```

Queries the available RF paths of the selected signal generators.

param list_py
'Path#1','Paht#2',...

set_select(*select: str*) → None

```
# SCPI: CPANel:UNUSed:PATH:SELect
driver.cpanel.unused.path.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

5.5.6 Used

SCPI Commands :

```
CPANel:USED:FREQuency
CPANel:USED:LEVel
CPANel:USED:LIST
CPANel:USED:SELect
CPANel:USED:STATe
```

class UsedCls

Used commands group definition. 7 total commands, 1 Subgroups, 5 group commands

get_frequency() → float

```
# SCPI: CPANel:USED:FREQuency
value: float = driver.cpanel.used.get_frequency()
```

Sets the RF frequency.

return
frequency: float

get_level() → float

```
# SCPI: CPANel:USED:LEVel
value: float = driver.cpanel.used.get_level()
```

Sets the RF level.

return
level: float

get_list_py() → List[str]

```
# SCPI: CPANel:USED:LIST
value: List[str] = driver.cpanel.used.get_list_py()
```

Queries the names of the used/unused signal generators.

return
list_py: 'Instr#1','Instr#2',...

get_select() → str

```
# SCPI: CPANel:USED:SELect
value: str = driver.cpanel.used.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

get_state() → bool

```
# SCPI: CPANel:USED:STATe
value: bool = driver.cpanel.used.get_state()
```

Activates the RF output of the selected signal generator.

return
state: ON| OFF| 1| 0

set_frequency(*frequency: float*) → None

```
# SCPI: CPANel:USED:FREQuency
driver.cpanel.used.set_frequency(frequency = 1.0)
```

Sets the RF frequency.

param frequency
float

set_level(*level: float*) → None

```
# SCPI: CPANel:USED:LEVel
driver.cpanel.used.set_level(level = 1.0)
```

Sets the RF level.

param level
float

set_select(*select: str*) → None

```
# SCPI: CPANel:USED:SELect
driver.cpanel.used.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_state(*state: bool*) → None

```
# SCPI: CPANel:USED:STATe
driver.cpanel.used.set_state(state = False)
```

Activates the RF output of the selected signal generator.

param state
ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.cpanel.used.clone()
```

Subgroups

5.5.6.1 Path

SCPI Commands :

```
CPANel:USED:PATH:LIST
CPANel:USED:PATH:SELect
```

class PathCls

Path commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_list_py() → List[str]

```
# SCPI: CPANel:USED:PATH:LIST
value: List[str] = driver.cpanel.used.path.get_list_py()
```

Queries the available RF paths of the selected signal generators.

return
list_py: 'Path#1','Paht#2',...

get_select() → str

```
# SCPI: CPANel:USED:PATH:SELect
value: str = driver.cpanel.used.path.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_select(*select: str*) → None

```
# SCPI: CPANel:USED:PATH:SELect
driver.cpanel.used.path.set_select(select = 'abc')
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

5.6 Destination

SCPI Commands :

```
DESTination:ADD
DESTination:CLEar
DESTination:COUNt
DESTination:DELete
DESTination:NAME
DESTination:SELect
```

class DestinationCls

Destination commands group definition. 12 total commands, 1 Subgroups, 6 group commands

clear() → None

```
# SCPI: DESTination:CLEar
driver.destination.clear()
```

Deletes all destinations from the current list.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: DESTination:CLEar
driver.destination.clear_with_opc()
```

Deletes all destinations from the current list.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: DESTination:DELete
driver.destination.delete(delete = 1.0)
```

Deletes the selected destination from the list.

param delete
float

get_count() → float

```
# SCPI: DESTination:COUNt
value: float = driver.destination.get_count()
```

Queries the number of available destinations.

return
count: integer

get_name() → str

```
# SCPI: DESTination:NAME
value: str = driver.destination.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → float

```
# SCPI: DESTination:SELect
value: float = driver.destination.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_add(*add: str*) → None

```
# SCPI: DESTination:ADD
driver.destination.set_add(add = 'abc')
```

Adds a destination to the list.

param add
string

set_name(*name: str*) → None

```
# SCPI: DESTination:NAME
driver.destination.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_select(*select: float*) → None

```
# SCPI: DESTination:SELect
driver.destination.set_select(select = 1.0)
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.destination.clone()
```

Subgroups

5.6.1 Plugin

SCPI Command :

```
DESTination:PLUGin:NAME
```

class PluginCls

Plugin commands group definition. 6 total commands, 1 Subgroups, 1 group commands

get_name() → str

```
# SCPI: DESTination:PLUGin:NAME
value: str = driver.destination.plugin.get_name()
```

Sets the name of the export plug-in.

return
name: string

set_name(*name: str*) → None

```
# SCPI: DESTination:PLUGin:NAME
driver.destination.plugin.set_name(name = 'abc')
```

Sets the name of the export plug-in.

param name
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.destination.plugin.clone()
```

Subgroups

5.6.1.1 Variable

SCPI Commands :

```
DESTination:PLUGin:VARiable:CATalog
DESTination:PLUGin:VARiable:RESet
DESTination:PLUGin:VARiable:VALue
```

class VariableCls

Variable commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_catalog() → str

```
# SCPI: DESTination:PLUGin:VARiable:CATalog
value: str = driver.destination.plugin.variable.get_catalog()
```

Queries the variables used in the plugin.

return
catalog: string

get_value() → str

```
# SCPI: DESTination:PLUGin:VARiable:VALue
value: str = driver.destination.plugin.variable.get_value()
```

Sets the values of the selected variable.

return
value: string

reset() → None

```
# SCPI: DESTination:PLUGin:VARiable:RESet
driver.destination.plugin.variable.reset()
```

Resets the variable values to the defaults.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: DESTination:PLUGin:VARiable:RESet
driver.destination.plugin.variable.reset_with_opc()
```

Resets the variable values to the defaults.

Same as reset, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_catalog(*catalog: str*) → None

```
# SCPI: DESTination:PLUGin:VARiable:CATalog
driver.destination.plugin.variable.set_catalog(catalog = 'abc')
```

Queries the variables used in the plugin.

param catalog
string

set_value(*value: str*) → None

```
# SCPI: DESTination:PLUGin:VARiable:VALue
driver.destination.plugin.variable.set_value(value = 'abc')
```

Sets the values of the selected variable.

param value
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.destination.plugin.variable.clone()
```

Subgroups

5.6.1.1.1 Select

SCPI Commands :

```
DESTination:PLUGin:VARiable:SELect:ID
DESTination:PLUGin:VARiable:SELect
```

class SelectCls

Select commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_id() → float

```
# SCPI: DESTination:PLUGin:VARiable:SELect:ID
value: float = driver.destination.plugin.variable.select.get_id()
```

Selects a plugin variable ID.

return
idn: float

get_value() → str

```
# SCPI: DESTination:PLUGin:VARiable:SELect
value: str = driver.destination.plugin.variable.select.get_value()
```

Selects a plugin variable.

return
select: string

set_id(*idn: float*) → None

```
# SCPI: DESTination:PLUGin:VARiable:SELect:ID
driver.destination.plugin.variable.select.set_id(idn = 1.0)
```

Selects a plugin variable ID.

param idn
float

set_value(*select: str*) → None

```
# SCPI: DESTination:PLUGin:VARiable:SELect
driver.destination.plugin.variable.select.set_value(select = 'abc')
```

Selects a plugin variable.

param select
string

5.7 Dialog

class DialogCls

Dialog commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.dialog.clone()
```

Subgroups

5.7.1 IpmPlot

SCPI Commands :

```
DIALog:IPMPlot:SAMPles
DIALog:IPMPlot:VIEW
```

class IpmPlotCls

IpmPlot commands group definition. 2 total commands, 0 Subgroups, 2 group commands

set_samples(*samples: float*) → None

```
# SCPI: DIALog:IPMPlot:SAMPles
driver.dialog.ipmPlot.set_samples(samples = 1.0)
```

Sets the number of values to be displayed in the preview diagram of the IPM profile.

param samples
float

set_view(*view: IpmPlotView*) → None

```
# SCPI: DIALog:IPMPlot:VIEW
driver.dialog.ipmPlot.set_view(view = enums.IpmPlotView.HISTogram)
```

Defines what kind of information is represented in the IPM profile diagram.

param view
TIMeseries| HISTogram TIMeseries Visualization of the profile variation over time HISTogram Statistical representation of the relative frequency density

5.8 Dsrc

SCPI Commands :

```
DSRC:CATalog
DSRC:CLEar
DSRC:COMMent
DSRC:CREate
DSRC:NAME
DSRC:REMove
DSRC:SELect
```

class DsrcCls

Dsrc commands group definition. 17 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: DSRC:CLEar
driver.dsrc.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: DSRC:CLEar
driver.dsrc.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_catalog() → str

```
# SCPI: DSRC:CATalog
value: str = driver.dsrc.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: DSRC:COMMent
value: str = driver.dsrc.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_name() → str

```
# SCPI: DSRC:NAME
value: str = driver.dsrc.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: DSRC:SELect
value: str = driver.dsrc.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_comment(*comment: str*) → None

```
# SCPI: DSRC:COMMent
driver.dsrc.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: DSRC:CREate
driver.dsrc.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: DSRC:NAME
driver.dsrc.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: DSRC:REMove
driver.dsrc.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: DSRC:SELect
driver.dsrc.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.dsrc.clone()
```

Subgroups

5.8.1 Item

SCPI Commands :

```
DSRC:ITEM:BITS
DSRC:ITEM:DATA
DSRC:ITEM:DELete
DSRC:ITEM:PATTern
DSRC:ITEM:SELect
DSRC:ITEM:TYPE
```

class ItemCls

Item commands group definition. 10 total commands, 2 Subgroups, 6 group commands

delete(*delete: float*) → None

```
# SCPI: DSRC:ITEM:DELete
driver.dsrc.item.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_bits() → float

```
# SCPI: DSRC:ITEM:BITS
value: float = driver.dsrc.item.get_bits()
```

Sets the length of the selected item in bits.

return
bits: float Range: 0 to 4096

get_data() → str

```
# SCPI: DSRC:ITEM:DATA
value: str = driver.dsrc.item.get_data()
```

Sets the user defined data pattern.

return
data: string

get_pattern() → ItemPattern

```
# SCPI: DSRC:ITEM:PATTern
value: enums.ItemPattern = driver.dsrc.item.get_pattern()
```

Sets the data pattern of the selected item.

return
pattern: ZERO| ONE| ALT| R2A| R2B| R3| R4A| R4B| R5| R7| R11| R13 ZERO|ONE Binary 0 and 1 ALT Variable bit strings ('1010') with alternating 0 and 1 and a maximum length of 999 bits R2A|R2B|R3|R4A|R4B|R5|R7|R11|R13 Barker codes

get_select() → float

```
# SCPI: DSRC:ITEM:SELect
value: float = driver.dsrc.item.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_type_py() → ItemTypeB

```
# SCPI: DSRC:ITEM:TYPE
value: enums.ItemTypeB = driver.dsrc.item.get_type_py()
```

Sets the data type of selected item.

return
type_py: PATTern| PRBS| USER

set_bits(*bits: float*) → None

```
# SCPI: DSRC:ITEM:BITS
driver.dsrc.item.set_bits(bits = 1.0)
```

Sets the length of the selected item in bits.

param bits
float Range: 0 to 4096

set_data(*data: str*) → None

```
# SCPI: DSRC:ITEM:DATA
driver.dsrc.item.set_data(data = 'abc')
```

Sets the user defined data pattern.

param data
string

set_pattern(*pattern: ItemPattern*) → None

```
# SCPI: DSRC:ITEM:PATTern
driver.dsrc.item.set_pattern(pattern = enums.ItemPattern.ALT)
```

Sets the data pattern of the selected item.

param pattern
ZERO| ONE| ALT| R2A| R2B| R3| R4A| R4B| R5| R7| R11| R13 ZERO|ONE Binary 0 and 1 ALT Variable bit strings ('1010') with alternating 0 and 1 and a maximum length of 999 bits R2A|R2B|R3|R4A|R4B|R5|R7|R11|R13 Barker codes

set_select(*select: float*) → None

```
# SCPI: DSRC:ITEM:SELect
driver.dsrc.item.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_type_py(*type_py: ItemTypeB*) → None

```
# SCPI: DSRC:ITEM:TYPE
driver.dsrc.item.set_type_py(type_py = enums.ItemTypeB.PATTern)
```

Sets the data type of selected item.

param type_py
PATTern| PRBS| USER

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.dsrc.item.clone()
```

Subgroups

5.8.1.1 Add

SCPI Command :

```
DSRC:ITEM:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: DSRC:ITEM:ADD
driver.dsrc.item.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: DSRC:ITEM:ADD
driver.dsrc.item.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.8.1.2 Prbs

SCPI Command :

```
DSRC:ITEM:PRBS:TYPE
```

class PrbsCls

Prbs commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_type_py() → PrbsType

```
# SCPI: DSRC:ITEM:PRBS:TYPE
value: enums.PrbsType = driver.dsrc.item.prbs.get_type_py()
```

Sets the PRBS type for the selected item.

return
type_py: P9| P11| P15| P16| P20| P21| P23| P7

set_type_py(*type_py: PrbsType*) → None

```
# SCPI: DSRC:ITEM:PRBS:TYPE
driver.dsrc.item.prbs.set_type_py(type_py = enums.PrbsType.P11)
```

Sets the PRBS type for the selected item.

param type_py
P9| P11| P15| P16| P20| P21| P23| P7

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.dsrc.item.prbs.clone()
```

Subgroups

5.8.1.2.1 Init

SCPI Commands :

```
DSRC:ITEM:PRBS:INIT:VALue
DSRC:ITEM:PRBS:INIT
```

class InitCls

Init commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_float_value() → float

```
# SCPI: DSRC:ITEM:PRBS:INIT:VALue
value: float = driver.dsrc.item.prbs.init.get_float_value()
```

Set a new initialization value.

return
value: float Range: 1 to 511

get_value() → bool

```
# SCPI: DSRC:ITEM:PRBS:INIT
value: bool = driver.dsrc.item.prbs.init.get_value()
```

Enables/disables initialization of the shift register with a user-defined value.

return
init: ON| OFF

set_float_value(*value: float*) → None

```
# SCPI: DSRC:ITEM:PRBS:INIT:VALue
driver.dsrc.item.prbs.init.set_float_value(value = 1.0)
```

Set a new initialization value.

param value
float Range: 1 to 511

set_value(*init: bool*) → None

```
# SCPI: DSRC:ITEM:PRBS:INIT
driver.dsrc.item.prbs.init.set_value(init = False)
```

Enables/disables initialization of the shift register with a user-defined value.

param init
ON| OFF

5.9 Emitter

SCPI Commands :

```
EMITter:CATalog
EMITter:COMMent
EMITter:CREate
EMITter:EIRP
EMITter:FREQuency
EMITter:NAME
EMITter:REMove
EMITter:SELect
```

class EmitterCls

Emitter commands group definition. 30 total commands, 1 Subgroups, 8 group commands

get_catalog() → str

```
# SCPI: EMITter:CATalog
value: str = driver.emitter.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: EMITter:COMMent
value: str = driver.emitter.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_eirp() → float

```
# SCPI: EMITter:EIRP
value: float = driver.emitter.get_eirp()
```

Sets the EIRP of the emitter.

return
eirp: float Range: -100 to 200, Unit: dBW

get_frequency() → float

```
# SCPI: EMITter:FREQuency
value: float = driver.emitter.get_frequency()
```

Sets the operating frequency.

return
frequency: float Range: 1000 to 1e+11, Unit: Hz

get_name() → str

```
# SCPI: EMITter:NAME
value: str = driver.emitter.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: EMITter:SELect
value: str = driver.emitter.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_comment(*comment: str*) → None

```
# SCPI: EMITter:COMMent
driver.emitter.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: EMITter:CREate
driver.emitter.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_eirp(*eirp: float*) → None

```
# SCPI: EMITter:EIRP
driver.emitter.set_eirp(eirp = 1.0)
```

Sets the EIRP of the emitter.

param eirp
float Range: -100 to 200, Unit: dBW

set_frequency(*frequency: float*) → None

```
# SCPI: EMITter:FREQuency
driver.emitter.set_frequency(frequency = 1.0)
```

Sets the operating frequency.

param frequency
float Range: 1000 to 1e+11, Unit: Hz

set_name(*name: str*) → None

```
# SCPI: EMITter:NAME
driver.emitter.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: EMITter:REMove
driver.emitter.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: EMITter:SELect
driver.emitter.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.emitter.clone()
```

Subgroups

5.9.1 Mode

SCPI Commands :

```
EMITter:MODE:ID
EMITter:MODE:CLEar
EMITter:MODE:COUNt
EMITter:MODE:DELete
EMITter:MODE:NAME
EMITter:MODE:SELect
```

class ModeCls

Mode commands group definition. 22 total commands, 4 Subgroups, 6 group commands

clear() → None

```
# SCPI: EMITter:MODE:CLEar
driver.emitter.mode.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: EMITter:MODE:CLEar
driver.emitter.mode.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: EMITter:MODE:DELete
driver.emitter.mode.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: EMITter:MODE:COUNt
value: float = driver.emitter.mode.get_count()
```

Queries the number of existing items.

return
count: integer

get_id() → float

```
# SCPI: EMITter:MODE:ID
value: float = driver.emitter.mode.get_id()
```

No command help available

return
idn: float Range: 1 to 65536

get_name() → str

```
# SCPI: EMITter:MODE:NAME
value: str = driver.emitter.mode.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → float

```
# SCPI: EMITter:MODE:SELect
value: float = driver.emitter.mode.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_id(*idn: float*) → None

```
# SCPI: EMITter:MODE:ID
driver.emitter.mode.set_id(idn = 1.0)
```

No command help available

param idn
float Range: 1 to 65536

set_name(*name: str*) → None

```
# SCPI: EMITter:MODE:NAME
driver.emitter.mode.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_select(*select: float*) → None

```
# SCPI: EMITter:MODE:SELect
driver.emitter.mode.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.emitter.mode.clone()
```

Subgroups

5.9.1.1 Add

SCPI Command :

```
EMITter:MODE:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: EMITter:MODE:ADD
driver.emitter.mode.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: EMITter:MODE:ADD
driver.emitter.mode.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.9.1.2 Antenna

SCPI Commands :

```
EMITter:MODE:ANTenna:CLEar
EMITter:MODE:ANTenna
```

class AntennaCls

Antenna commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: EMITter:MODE:ANTenna:CLEar
driver.emitter.mode.antenna.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: EMITter:MODE:ANTenna:CLEar
driver.emitter.mode.antenna.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: EMITter:MODE:ANTenna
value: str = driver.emitter.mode.antenna.get_value()
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

return
antenna: string

set_value(*antenna: str*) → None

```
# SCPI: EMITter:MODE:ANTenna
driver.emitter.mode.antenna.set_value(antenna = 'abc')
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

param antenna
string

5.9.1.3 Beam

SCPI Commands :

```
EMITter:MODE:BEAM:CLEar
EMITter:MODE:BEAM:COUNt
EMITter:MODE:BEAM:DELete
EMITter:MODE:BEAM:NAME
EMITter:MODE:BEAM:SELect
EMITter:MODE:BEAM:SEQuence
EMITter:MODE:BEAM:STATe
```

class BeamCls

Beam commands group definition. 11 total commands, 2 Subgroups, 7 group commands

clear() → None

```
# SCPI: EMITter:MODE:BEAM:CLEar
driver.emitter.mode.beam.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: EMITter:MODE:BEAM:CLEar
driver.emitter.mode.beam.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: EMITter:MODE:BEAM:DELete
driver.emitter.mode.beam.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: EMITter:MODE:BEAM:COUNt
value: float = driver.emitter.mode.beam.get_count()
```

Queries the number of existing items.

return
count: integer

get_name() → str

```
# SCPI: EMITter:MODE:BEAM:NAME
value: str = driver.emitter.mode.beam.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → float

```
# SCPI: EMITter:MODE:BEAM:SELect
value: float = driver.emitter.mode.beam.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_sequence() → str

```
# SCPI: EMITter:MODE:BEAM:SEQuence
value: str = driver.emitter.mode.beam.get_sequence()
```

Assigns a pulse sequence, see method RsPulseSeq.Sequence.create.

return
sequence: string

get_state() → bool

```
# SCPI: EMITter:MODE:BEAM:STATe
value: bool = driver.emitter.mode.beam.get_state()
```

Activates a beam.

return
state: ON| OFF| 1| 0

set_name(*name: str*) → None

```
# SCPI: EMITter:MODE:BEAM:NAME
driver.emitter.mode.beam.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_select(*select: float*) → None

```
# SCPI: EMITter:MODE:BEAM:SELect
driver.emitter.mode.beam.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to …:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_sequence(*sequence: str*) → None

```
# SCPI: EMITter:MODE:BEAM:SEQuence
driver.emitter.mode.beam.set_sequence(sequence = 'abc')
```

Assigns a pulse sequence, see method RsPulseSeq.Sequence.create.

param sequence
string

set_state(*state: bool*) → None

```
# SCPI: EMITter:MODE:BEAM:STATe
driver.emitter.mode.beam.set_state(state = False)
```

Activates a beam.

param state
ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.emitter.mode.beam.clone()
```

Subgroups

5.9.1.3.1 Add

SCPI Command :

```
EMITter:MODE:BEAM:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: EMITter:MODE:BEAM:ADD
driver.emitter.mode.beam.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: EMITter:MODE:BEAM:ADD
driver.emitter.mode.beam.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.9.1.3.2 Offset

SCPI Commands :

```
EMITter:MODE:BEAM:OFFSet:AZIMuth
EMITter:MODE:BEAM:OFFSet:ELEVation
EMITter:MODE:BEAM:OFFSet:FREQuency
```

class OffsetCls

Offset commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_azimuth() → float

```
# SCPI: EMITter:MODE:BEAM:OFFSet:AZIMuth
value: float = driver.emitter.mode.beam.offset.get_azimuth()
```

Sets the Azimuth value for the beam offset.

return
azimuth: float Range: 0 to 360

get_elevation() → float

```
# SCPI: EMITter:MODE:BEAM:OFFSet:ELEVation
value: float = driver.emitter.mode.beam.offset.get_elevation()
```

Offsets the position of the beam in both the azimuth or elevation.

return
elevation: float Range: -90 to 90

get_frequency() → float

```
# SCPI: EMITter:MODE:BEAM:OFFSet:FREQuency
value: float = driver.emitter.mode.beam.offset.get_frequency()
```

Offsets the frequency of the beam.

return
frequency: float Range: -1e+09 to 1e+09

set_azimuth(*azimuth: float*) → None

```
# SCPI: EMITter:MODE:BEAM:OFFSet:AZIMuth
driver.emitter.mode.beam.offset.set_azimuth(azimuth = 1.0)
```

Sets the Azimuth value for the beam offset.

param azimuth
float Range: 0 to 360

set_elevation(*elevation: float*) → None

```
# SCPI: EMITter:MODE:BEAM:OFFSet:ELEVation
driver.emitter.mode.beam.offset.set_elevation(elevation = 1.0)
```

Offsets the position of the beam in both the azimuth or elevation.

param elevation
float Range: -90 to 90

set_frequency(*frequency: float*) → None

```
# SCPI: EMITter:MODE:BEAM:OFFSet:FREQuency
driver.emitter.mode.beam.offset.set_frequency(frequency = 1.0)
```

Offsets the frequency of the beam.

param frequency
float Range: -1e+09 to 1e+09

5.9.1.4 Scan

SCPI Commands :

```
EMITter:MODE:SCAN:CLEar
EMITter:MODE:SCAN
```

class ScanCls

Scan commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: EMITter:MODE:SCAN:CLEar
driver.emitter.mode.scan.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: EMITter:MODE:SCAN:CLEar
driver.emitter.mode.scan.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: EMITter:MODE:SCAN
value: str = driver.emitter.mode.scan.get_value()
```

Assigns an antenna scan, see method RsPulseSeq.Scan.create.

return
scan: string

set_value(*scan: str*) → None

```
# SCPI: EMITter:MODE:SCAN
driver.emitter.mode.scan.set_value(scan = 'abc')
```

Assigns an antenna scan, see method RsPulseSeq.Scan.create.

param scan
string

5.10 Generator

SCPI Command :

```
GENerator:TYPE
```

class GeneratorCls

Generator commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_type_py() → GeneratorType

```
# SCPI: GENerator:TYPE
value: enums.GeneratorType = driver.generator.get_type_py()
```

Sets the generator type.

return
type_py: SMBB| SW| SGT| SMBV| SMJ| SMW| SMU| SMM SMBB designates R&S SMBV100B.

set_type_py(*type_py: GeneratorType*) → None

```
# SCPI: GENerator:TYPE
driver.generator.set_type_py(type_py = enums.GeneratorType.SGT)
```

Sets the generator type.

param type_py
SMBB| SW| SGT| SMBV| SMJ| SMW| SMU| SMM SMBB designates R&S SMBV100B.

5.11 ImportPy

class ImportPyCls

ImportPy commands group definition. 50 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.importPy.clone()
```

Subgroups

5.11.1 Pdw

SCPI Commands :

```
IMPort:PDW:NORM
IMPort:PDW:STATus
```

class PdwCls

Pdw commands group definition. 44 total commands, 4 Subgroups, 2 group commands

get_norm() → bool

```
# SCPI: IMPort:PDW:NORM
value: bool = driver.importPy.pdw.get_norm()
```

Normalizes the TOA (time of arrival) of the first pulse to 0. Subsequent TOAs are relative.

return
norm: ON| OFF| 1| 0

get_status() → bool

```
# SCPI: IMPort:PDW:STATus
value: bool = driver.importPy.pdw.get_status()
```

Queries the parsing status.

return
status: ON| OFF| 1| 0 1 Import completed

set_norm(*norm: bool*) → None

```
# SCPI: IMPort:PDW:NORM
driver.importPy.pdw.set_norm(norm = False)
```

Normalizes the TOA (time of arrival) of the first pulse to 0. Subsequent TOAs are relative.

param norm
ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.importPy.pdw.clone()
```

Subgroups

5.11.1.1 Data

SCPI Commands :

```
IMPort:PDW:DATA:FREQuency
IMPort:PDW:DATA:LEVel
IMPort:PDW:DATA:MOP
IMPort:PDW:DATA:OFFSet
IMPort:PDW:DATA:PHASe
IMPort:PDW:DATA:SEL
IMPort:PDW:DATA:TOA
IMPort:PDW:DATA:WIDTh
```

class DataCls

Data commands group definition. 34 total commands, 9 Subgroups, 8 group commands

get_frequency() → float

```
# SCPI: IMPort:PDW:DATA:FREQuency
value: float = driver.importPy.pdw.data.get_frequency()
```

Queries the pulse parameter.

return
frequency: float

get_level() → float

```
# SCPI: IMPort:PDW:DATA:LEVel
value: float = driver.importPy.pdw.data.get_level()
```

Queries the pulse parameter.

return
level: float

get_mop() → DataMop

```
# SCPI: IMPort:PDW:DATA:MOP
value: enums.DataMop = driver.importPy.pdw.data.get_mop()
```

Queries the used modulation on pulse (MOP) . Use the corresponding command to query further pulse and modulation parameter for the respective MOP.

return
mop: CW| AM| FM| ASK| FSK| PSK| LFM| NLFM| TFM| BKR2a| BKR2b| BKR3| BKR4a| BKR4b| BKR5| BKR7| BKR11| BKR13| CPH| PLFM

get_offset() → float

```
# SCPI: IMPort:PDW:DATA:OFFSet
value: float = driver.importPy.pdw.data.get_offset()
```

Queries the pulse parameter.

return
offset: float

get_phase() → float

```
# SCPI: IMPort:PDW:DATA:PHASe
value: float = driver.importPy.pdw.data.get_phase()
```

Queries the pulse parameter.

return
phase: float

get_sel() → float

```
# SCPI: IMPort:PDW:DATA:SEL
value: float = driver.importPy.pdw.data.get_sel()
```

Selects the pulse for that the further queries apply.

return
sel: float Range: 1 to max

get_toa() → float

```
# SCPI: IMPort:PDW:DATA:TOA
value: float = driver.importPy.pdw.data.get_toa()
```

Queries the pulse parameter.

return
toa: float

get_width() → float

```
# SCPI: IMPort:PDW:DATA:WIDTh
value: float = driver.importPy.pdw.data.get_width()
```

Queries the pulse parameter.

return
width: float

set_sel(*sel: float*) → None

```
# SCPI: IMPort:PDW:DATA:SEL
driver.importPy.pdw.data.set_sel(sel = 1.0)
```

Selects the pulse for that the further queries apply.

param sel
float Range: 1 to max

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.importPy.pdw.data.clone()
```

Subgroups

5.11.1.1.1 Am

SCPI Commands :

```
IMPort:PDW:DATA:AM:DEPTh
IMPort:PDW:DATA:AM:MODFreq
```

class AmCls

Am commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_depth() → float

```
# SCPI: IMPort:PDW:DATA:AM:DEPTh
value: float = driver.importPy.pdw.data.am.get_depth()
```

Queries the pulse parameter.

return
depth: float

get_mod_freq() → float

```
# SCPI: IMPort:PDW:DATA:AM:MODFreq
value: float = driver.importPy.pdw.data.am.get_mod_freq()
```

Queries the pulse parameter.

return
mod_freq: float

5.11.1.1.2 Ask

SCPI Commands :

```
IMPort:PDW:DATA:ASK:CHIPcount
IMPort:PDW:DATA:ASK:PATTern
IMPort:PDW:DATA:ASK:RATE
IMPort:PDW:DATA:ASK:STATes
IMPort:PDW:DATA:ASK:STEP
```

class AskCls

Ask commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_chip_count() → float

```
# SCPI: IMPort:PDW:DATA:ASK:CHIPcount
value: float = driver.importPy.pdw.data.ask.get_chip_count()
```

Queries the pulse parameter.

return
chip_count: float

get_pattern() → str

```
# SCPI: IMPort:PDW:DATA:ASK:PATTern
value: str = driver.importPy.pdw.data.ask.get_pattern()
```

Queries the pulse parameter.

return
pattern: string

get_rate() → float

```
# SCPI: IMPort:PDW:DATA:ASK:RATE
value: float = driver.importPy.pdw.data.ask.get_rate()
```

Queries the pulse parameter.

return
rate: float

get_states() → float

```
# SCPI: IMPort:PDW:DATA:ASK:STATes
value: float = driver.importPy.pdw.data.ask.get_states()
```

Queries the pulse parameter.

return
states: float

get_step() → float

```
# SCPI: IMPort:PDW:DATA:ASK:STEP
value: float = driver.importPy.pdw.data.ask.get_step()
```

Queries the pulse parameter.

return
step: float

5.11.1.1.3 Cph

SCPI Commands :

```
IMPort:PDW:DATA:CPH:CHIPcount
IMPort:PDW:DATA:CPH:VALues
```

class CphCls

Cph commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_chip_count() → float

```
# SCPI: IMPort:PDW:DATA:CPH:CHIPcount
value: float = driver.importPy.pdw.data.cph.get_chip_count()
```

Queries the pulse parameter.

return
chip_count: float

get_values() → str

```
# SCPI: IMPort:PDW:DATA:CPH:VALues
value: str = driver.importPy.pdw.data.cph.get_values()
```

Queries the pulse parameter.

return
values: float

set_values(*values: str*) → None

```
# SCPI: IMPort:PDW:DATA:CPH:VALues
driver.importPy.pdw.data.cph.set_values(values = 'abc')
```

Queries the pulse parameter.

param values
No help available

5.11.1.1.4 Fm

SCPI Commands :

```
IMPort:PDW:DATA:FM:DEViation
IMPort:PDW:DATA:FM:MODFreq
```

class FmCls

Fm commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_deviation() → float

```
# SCPI: IMPort:PDW:DATA:FM:DEViation
value: float = driver.importPy.pdw.data.fm.get_deviation()
```

Queries the pulse parameter.

return
deviation: float

get_mod_freq() → float

```
# SCPI: IMPort:PDW:DATA:FM:MODFreq
value: float = driver.importPy.pdw.data.fm.get_mod_freq()
```

Queries the pulse parameter.

return
mod_freq: float

5.11.1.1.5 Fsk

SCPI Commands :

```
IMPort:PDW:DATA:FSK:CHIPcount
IMPort:PDW:DATA:FSK:PATTern
IMPort:PDW:DATA:FSK:RATE
IMPort:PDW:DATA:FSK:STATes
IMPort:PDW:DATA:FSK:STEP
```

class FskCls

Fsk commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_chip_count() → float

```
# SCPI: IMPort:PDW:DATA:FSK:CHIPcount
value: float = driver.importPy.pdw.data.fsk.get_chip_count()
```

Queries the pulse parameter.

return
chip_count: float

get_pattern() → str

```
# SCPI: IMPort:PDW:DATA:FSK:PATTern
value: str = driver.importPy.pdw.data.fsk.get_pattern()
```

Queries the pulse parameter.

return
pattern: string

get_rate() → float

```
# SCPI: IMPort:PDW:DATA:FSK:RATE
value: float = driver.importPy.pdw.data.fsk.get_rate()
```

Queries the pulse parameter.

return
rate: float

get_states() → float

```
# SCPI: IMPort:PDW:DATA:FSK:STATes
value: float = driver.importPy.pdw.data.fsk.get_states()
```

Queries the pulse parameter.

return
states: float

get_step() → float

```
# SCPI: IMPort:PDW:DATA:FSK:STEP
value: float = driver.importPy.pdw.data.fsk.get_step()
```

Queries the pulse parameter.

return
step: float

5.11.1.1.6 Lfm

SCPI Command :

```
IMPort:PDW:DATA:LFM:RATE
```

class LfmCls

Lfm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_rate() → float

```
# SCPI: IMPort:PDW:DATA:LFM:RATE
value: float = driver.importPy.pdw.data.lfm.get_rate()
```

Queries the pulse parameter.

return
rate: float

5.11.1.1.7 NlFm

SCPI Commands :

```
IMPort:PDW:DATA:NLFM:CUBic
IMPort:PDW:DATA:NLFM:LINear
IMPort:PDW:DATA:NLFM:QUADratic
```

class NlFmCls

NlFm commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_cubic() → float

```
# SCPI: IMPort:PDW:DATA:NLFM:CUBic
value: float = driver.importPy.pdw.data.nlFm.get_cubic()
```

Queries the pulse parameter.

return
cubic: float

get_linear() → float

```
# SCPI: IMPort:PDW:DATA:NLFM:LINear
value: float = driver.importPy.pdw.data.nlFm.get_linear()
```

Queries the pulse parameter.

return
linear: float

get_quadratic() → float

```
# SCPI: IMPort:PDW:DATA:NLFM:QUADratic
value: float = driver.importPy.pdw.data.nlFm.get_quadratic()
```

Queries the pulse parameter.

return
quadratic: float

5.11.1.1.8 PlFm

SCPI Command :

```
IMPort:PDW:DATA:PLFM:VALues
```

class PlFmCls

PlFm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_values() → str

```
# SCPI: IMPort:PDW:DATA:PLFM:VALues
value: str = driver.importPy.pdw.data.plFm.get_values()
```

Queries the pulse parameter.

return
values: float

set_values(*values: str*) → None

```
# SCPI: IMPort:PDW:DATA:PLFM:VALues
driver.importPy.pdw.data.plFm.set_values(values = 'abc')
```

Queries the pulse parameter.

param values
No help available

5.11.1.1.9 Psk

SCPI Commands :

```
IMPort:PDW:DATA:PSK:CHIPcount
IMPort:PDW:DATA:PSK:PATTern
IMPort:PDW:DATA:PSK:RATE
IMPort:PDW:DATA:PSK:STATes
IMPort:PDW:DATA:PSK:STEP
```

class PskCls

Psk commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_chip_count() → float

```
# SCPI: IMPort:PDW:DATA:PSK:CHIPcount
value: float = driver.importPy.pdw.data.psk.get_chip_count()
```

Queries the pulse parameter.

return
chip_count: float

get_pattern() → str

```
# SCPI: IMPort:PDW:DATA:PSK:PATTern
value: str = driver.importPy.pdw.data.psk.get_pattern()
```

Queries the pulse parameter.

return
pattern: string

get_rate() → float

```
# SCPI: IMPort:PDW:DATA:PSK:RATE
value: float = driver.importPy.pdw.data.psk.get_rate()
```

Queries the pulse parameter.

return
rate: float

get_states() → float

```
# SCPI: IMPort:PDW:DATA:PSK:STATes
value: float = driver.importPy.pdw.data.psk.get_states()
```

Queries the pulse parameter.

return
states: float

get_step() → float

```
# SCPI: IMPort:PDW:DATA:PSK:STEP
value: float = driver.importPy.pdw.data.psk.get_step()
```

Queries the pulse parameter.

return
step: float

5.11.1.2 Execute

SCPI Command :

```
IMPort:PDW:EXECute
```

class ExecuteCls

Execute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: IMPort:PDW:EXECute
driver.importPy.pdw.execute.set()
```

Starts parsing the PDW list file.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:PDW:EXECute
driver.importPy.pdw.execute.set_with_opc()
```

Starts parsing the PDW list file.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.11.1.3 File

class FileCls

File commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.importPy.pdw.file.clone()
```

Subgroups

5.11.1.3.1 Pdw

SCPI Commands :

```
IMPort:PDW:FILE:PDW:LOAD
IMPort:PDW:FILE:PDW:SAVE
IMPort:PDW:FILE:PDW
```

class PdwCls

Pdw commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_value() → str

```
# SCPI: IMPort:PDW:FILE:PDW
value: str = driver.importPy.pdw.file.pdw.get_value()
```

Sets or queries the name of the used PDW list file.

return
pdw: absolute file path and filename, incl. file extension

load() → None

```
# SCPI: IMPort:PDW:FILE:PDW:LOAD
driver.importPy.pdw.file.pdw.load()
```

Loads the selected file.

load_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:PDW:FILE:PDW:LOAD
driver.importPy.pdw.file.pdw.load_with_opc()
```

Loads the selected file.

Same as load, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

save() → None

```
# SCPI: IMPort:PDW:FILE:PDW:SAVE
driver.importPy.pdw.file.pdw.save()
```

Stores the selected file.

save_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:PDW:FILE:PDW:SAVE
driver.importPy.pdw.file.pdw.save_with_opc()
```

Stores the selected file.

Same as save, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

`set_value`(*pdw: str*) → None

```
# SCPI: IMPort:PDW:FILE:PDW
driver.importPy.pdw.file.pdw.set_value(pdw = 'abc')
```

Sets or queries the name of the used PDW list file.

param pdw
absolute file path and filename, incl. file extension

5.11.1.3.2 Template

SCPI Commands :

```
IMPort:PDW:FILE:TEMPlate:LOAD
IMPort:PDW:FILE:TEMPlate:SAVE
IMPort:PDW:FILE:TEMPlate
```

`class TemplateCls`

Template commands group definition. 3 total commands, 0 Subgroups, 3 group commands

`get_value`() → str

```
# SCPI: IMPort:PDW:FILE:TEMPlate
value: str = driver.importPy.pdw.file.template.get_value()
```

Sets or queries the name of the used import template file.

return
template: absolute file path and filename, incl. file extension

`load`() → None

```
# SCPI: IMPort:PDW:FILE:TEMPlate:LOAD
driver.importPy.pdw.file.template.load()
```

Loads the selected file.

`load_with_opc`(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:PDW:FILE:TEMPlate:LOAD
driver.importPy.pdw.file.template.load_with_opc()
```

Loads the selected file.

Same as load, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

save() → None

```
# SCPI: IMPort:PDW:FILE:TEMPlate:SAVE
driver.importPy.pdw.file.template.save()
```

Stores the selected file.

save_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:PDW:FILE:TEMPlate:SAVE
driver.importPy.pdw.file.template.save_with_opc()
```

Stores the selected file.

Same as save, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*template: str*) → None

```
# SCPI: IMPort:PDW:FILE:TEMPlate
driver.importPy.pdw.file.template.set_value(template = 'abc')
```

Sets or queries the name of the used import template file.

param template
absolute file path and filename, incl. file extension

5.11.1.4 Store

SCPI Command :

```
IMPort:PDW:STORe
```

class StoreCls

Store commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: IMPort:PDW:STORe
driver.importPy.pdw.store.set()
```

Stores the imported PDW list file as waveform element in the repository.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:PDW:STORe
driver.importPy.pdw.store.set_with_opc()
```

Stores the imported PDW list file as waveform element in the repository.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.11.2 View

SCPI Command :

```
IMPort:VIEW:COUNt
```

class ViewCls

View commands group definition. 6 total commands, 2 Subgroups, 1 group commands

get_count() → ViewCount

```
# SCPI: IMPort:VIEW:COUNt
value: enums.ViewCount = driver.importPy.view.get_count()
```

Sets the entries per page to be displayed.

return
count: 50| 100| 500| 1000| 5000| 10000| 50000| 100000

set_count(*count: ViewCount*) → None

```
# SCPI: IMPort:VIEW:COUNt
driver.importPy.view.set_count(count = enums.ViewCount._100)
```

Sets the entries per page to be displayed.

param count
50| 100| 500| 1000| 5000| 10000| 50000| 100000

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.importPy.view.clone()
```

Subgroups

5.11.2.1 Move

SCPI Command :

```
IMPort:VIEW:MOVE:STARt
```

class MoveCls

Move commands group definition. 4 total commands, 3 Subgroups, 1 group commands

start() → None

```
# SCPI: IMPort:VIEW:MOVE:STARt
driver.importPy.view.move.start()
```

Goes to the first/next/previous/last page.

start_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:VIEW:MOVE:STARt
driver.importPy.view.move.start_with_opc()
```

Goes to the first/next/previous/last page.

Same as start, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.importPy.view.move.clone()
```

Subgroups

5.11.2.1.1 Backwards

SCPI Command :

```
IMPort:VIEW:MOVE:BACKwards
```

class BackwardsCls

Backwards commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: IMPort:VIEW:MOVE:BACKwards
driver.importPy.view.move.backwards.set()
```

Goes to the first/next/previous/last page.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:VIEW:MOVE:BACKwards
driver.importPy.view.move.backwards.set_with_opc()
```

Goes to the first/next/previous/last page.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.11.2.1.2 End

SCPI Command :

```
IMPort:VIEW:MOVE:END
```

class EndCls

End commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: IMPort:VIEW:MOVE:END
driver.importPy.view.move.end.set()
```

Goes to the first/next/previous/last page.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:VIEW:MOVE:END
driver.importPy.view.move.end.set_with_opc()
```

Goes to the first/next/previous/last page.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.11.2.1.3 Forward

SCPI Command :

```
IMPort:VIEW:MOVE:FORWard
```

class ForwardCls

Forward commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: IMPort:VIEW:MOVE:FORWard
driver.importPy.view.move.forward.set()
```

Goes to the first/next/previous/last page.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IMPort:VIEW:MOVE:FORWard
driver.importPy.view.move.forward.set_with_opc()
```

Goes to the first/next/previous/last page.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.11.2.2 Time

SCPI Command :

```
IMPort:VIEW:TIME:STARt
```

class TimeCls

Time commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_start() → float

```
# SCPI: IMPort:VIEW:TIME:STARt
value: float = driver.importPy.view.time.get_start()
```

Sets the start line displayed on the page.

return
start: float

set_start(*start: float*) → None

```
# SCPI: IMPort:VIEW:TIME:STARt
driver.importPy.view.time.set_start(start = 1.0)
```

Sets the start line displayed on the page.

param start
float

5.12 Instrument

SCPI Commands :

```
INSTrument:ADD
INSTrument:CAPabilities
INSTrument:CLEar
INSTrument:COMMent
INSTrument:CONNect
INSTrument:COUNt
INSTrument:DELete
INSTrument:FIRMware
INSTrument:LIST
INSTrument:NAME
INSTrument:ONLine
INSTrument:PMOD
INSTrument:PSEC
INSTrument:RESource
INSTrument:SELect
INSTrument:SUPPorted
```

(continues on next page)

(continued from previous page)

```
INSTrument:TYPE
INSTrument:VIRTual
```

class InstrumentCls

Instrument commands group definition. 19 total commands, 1 Subgroups, 18 group commands

clear() → None

```
# SCPI: INSTrument:CLEar
driver.instrument.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INSTrument:CLEar
driver.instrument.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: INSTrument:DELete
driver.instrument.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_capabilities() → str

```
# SCPI: INSTrument:CAPabilities
value: str = driver.instrument.get_capabilities()
```

Queries the generator capabilities regarding supported scenario types and processing of waveforms and multi-segment files.

return
capabilities: string

get_comment() → str

```
# SCPI: INSTrument:COMMent
value: str = driver.instrument.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_count() → float

```
# SCPI: INSTrument:COUNt
value: float = driver.instrument.get_count()
```

Queries the number of existing items.

return
count: integer

get_firmware() → str

```
# SCPI: INSTrument:FIRMware
value: str = driver.instrument.get_firmware()
```

Queries the firmware version of the selected instrument.

return
firmware: string

get_list_py() → List[str]

```
# SCPI: INSTrument:LIST
value: List[str] = driver.instrument.get_list_py()
```

Queries the names of the signal generators that are connected to the R&S Pulse Sequencer in the current setup. See method RsPulseSeq.Setup.listPy.

return
list_py: 'Instr#1','Instr#2',...

get_name() → str

```
# SCPI: INSTrument:NAME
value: str = driver.instrument.get_name()
```

Queries the name of the selected instrument.

return
name: string

get_online() → float

```
# SCPI: INSTrument:ONLine
value: float = driver.instrument.get_online()
```

Queries the connection status of a physical signal generator.

return
online: float 0 Offline 1 Online

get_pmod() → PmodSource

```
# SCPI: INSTrument:PMOD
value: enums.PmodSource = driver.instrument.get_pmod()
```

Select the marker source if pulse modulator is used, see method RsPulseSeq.Setup.Pmod.enable.

return
pmod: OFF| INTernal| EXTernal OFF Disables the function for the selected instrument. INTernal No additional cabling or configuration of marker signals is required. EXTernal Requires that: 1) the instruments are cabled according to the wiring diagram 2) high the clock rate is selected 3) pulse marker M2 is set to pulse width 4) sequence marker M2 is enabled 5) the generation of the dedicated marker signal is enabled

get_psec() → Psec

```
# SCPI: INSTrument:PSEC
value: enums.Psec = driver.instrument.get_psec()
```

Sets the Primary/Secondary order in synchronized setups.

return
psec: NONE| PRIMary| SEC1| SEC2| SEC3| SEC4| SEC5| SEC6| SEC7| SEC8| SEC9| SEC10| SEC11| SEC12| SEC13| SEC14| SEC15| SEC16 NONE Unsynchronized instrument. PRIMary The primary instrument. SEC1 Denotes the secondary instruments.

get_resource() → str

```
# SCPI: INSTrument:RESource
value: str = driver.instrument.get_resource()
```

Queries the resource string of the instrument selected with the command method RsPulseSeq.Instrument.select.

return
resource: string

get_select() → float

```
# SCPI: INSTrument:SELect
value: float = driver.instrument.get_select()
```

Selects the element to which the subsequent commands apply.

return
select: string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

get_supported() → float

```
# SCPI: INSTrument:SUPPorted
value: float = driver.instrument.get_supported()
```

Queries the supported status.

return
supported: 0| 1 1 Supported 0 Not supported

get_type_py() → GeneratorType

```
# SCPI: INSTrument:TYPE
value: enums.GeneratorType = driver.instrument.get_type_py()
```

Sets the instrument type.

return
type_py: SMBB| SW| SGT| SMBV| SMJ| SMW| SMU| SMM

get_virtual() → bool

```
# SCPI: INSTrument:VIRTual
value: bool = driver.instrument.get_virtual()
```

Queries the state of the virtual instrument.

return
virtual: ON| OFF| 1| 0

set_add(*add: str*) → None

```
# SCPI: INSTrument:ADD
driver.instrument.set_add(add = 'abc')
```

Adds an instrument with the specified IP address, computer name, or the VISA resource string.

param add
string 'IP_Address' or 'Computer_Name' or 'GPIB_Address'

set_capabilities(*capabilities: str*) → None

```
# SCPI: INSTrument:CAPabilities
driver.instrument.set_capabilities(capabilities = 'abc')
```

Queries the generator capabilities regarding supported scenario types and processing of waveforms and multi-segment files.

param capabilities
string

set_comment(*comment: str*) → None

```
# SCPI: INSTrument:COMMent
driver.instrument.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_connect(*connect: str*) → None

```
# SCPI: INSTrument:CONNect
driver.instrument.set_connect(connect = 'abc')
```

Connects to the instrument defined by method RsPulseSeq.Instrument.select

param connect
string

set_name(*name: str*) → None

```
# SCPI: INSTrument:NAME
driver.instrument.set_name(name = 'abc')
```

Queries the name of the selected instrument.

param name
string

set_online(*online: float*) → None

```
# SCPI: INSTrument:ONLine
driver.instrument.set_online(online = 1.0)
```

Queries the connection status of a physical signal generator.

param online
float 0 Offline 1 Online

set_pmod(*pmod: PmodSource*) → None

```
# SCPI: INSTrument:PMOD
driver.instrument.set_pmod(pmod = enums.PmodSource.EXTernal)
```

Select the marker source if pulse modulator is used, see method RsPulseSeq.Setup.Pmod.enable.

param pmod
OFF| INTernal| EXTernal OFF Disables the function for the selected instrument. INTernal No additional cabling or configuration of marker signals is required. EXTernal Requires that: 1) the instruments are cabled according to the wiring diagram 2) high the clock rate is selected 3) pulse marker M2 is set to pulse width 4) sequence marker M2 is enabled 5) the generation of the dedicated marker signal is enabled

set_psec(*psec: Psec*) → None

```
# SCPI: INSTrument:PSEC
driver.instrument.set_psec(psec = enums.Psec.NONE)
```

Sets the Primary/Secondary order in synchronized setups.

param psec
NONE| PRIMary| SEC1| SEC2| SEC3| SEC4| SEC5| SEC6| SEC7| SEC8| SEC9| SEC10| SEC11| SEC12| SEC13| SEC14| SEC15| SEC16 NONE Unsynchronized instrument. PRIMary The primary instrument. SEC1 Denotes the secondary instruments.

set_select(*select: float*) → None

```
# SCPI: INSTrument:SELect
driver.instrument.set_select(select = 1.0)
```

Selects the element to which the subsequent commands apply.

param select
string Available element as queried with the corresponding ...:LIST command. For example, method RsPulseSeq.Assignment.Generator.Path.Antenna.listPy

set_type_py(*type_py: GeneratorType*) → None

```
# SCPI: INSTrument:TYPE
driver.instrument.set_type_py(type_py = enums.GeneratorType.SGT)
```

Sets the instrument type.

param type_py
SMBB| SW| SGT| SMBV| SMJ| SMW| SMU| SMM

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.instrument.clone()
```

Subgroups

5.12.1 Adb

SCPI Command :

```
INSTrument:ADB:STATe
```

class AdbCls

Adb commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_state() → bool

```
# SCPI: INSTrument:ADB:STATe
value: bool = driver.instrument.adb.get_state()
```

Defines the instrument that holds the adjustment data of hardware setup.

return
state: ON| OFF| 1| 0

set_state(*state: bool*) → None

```
# SCPI: INSTrument:ADB:STATe
driver.instrument.adb.set_state(state = False)
```

Defines the instrument that holds the adjustment data of hardware setup.

param state
ON| OFF| 1| 0

5.13 Ipm

SCPI Commands :

```
IPM:CATalog
IPM:COMMent
IPM:CREate
IPM:EQUation
IPM:NAME
IPM:REMove
IPM:SELect
```

(continues on next page)

(continued from previous page)

```
IPM:TYPE
IPM:UNIT
```

class IpmCls

Ipm commands group definition. 66 total commands, 9 Subgroups, 9 group commands

get_catalog() → str

```
# SCPI: IPM:CATalog
value: str = driver.ipm.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: IPM:COMMent
value: str = driver.ipm.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_equation() → str

```
# SCPI: IPM:EQUation
value: str = driver.ipm.get_equation()
```

Defines the IPM shape as a function.

return
equation: string

get_name() → str

```
# SCPI: IPM:NAME
value: str = driver.ipm.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: IPM:SELect
value: str = driver.ipm.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

get_type_py() → IpmType

```
# SCPI: IPM:TYPE
value: enums.IpmType = driver.ipm.get_type_py()
```

Sets the shape of the profile.

return
type_py: STEPs| WAVeform| RLISt| LIST| SHAPe| RANDom| EQUation| PLUGin| RSTep| BINomial

get_unit() → Units

```
# SCPI: IPM:UNIT
value: enums.Units = driver.ipm.get_unit()
```

Sets the units of the profile.

return
unit: NONE| SEConds| HERTz| DB| DEGRees| PERCent

set_comment(*comment: str*) → None

```
# SCPI: IPM:COMMent
driver.ipm.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: IPM:CREate
driver.ipm.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_equation(*equation: str*) → None

```
# SCPI: IPM:EQUation
driver.ipm.set_equation(equation = 'abc')
```

Defines the IPM shape as a function.

param equation
string

set_name(*name: str*) → None

```
# SCPI: IPM:NAME
driver.ipm.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: IPM:REMove
driver.ipm.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: IPM:SELect
driver.ipm.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_type_py(*type_py: IpmType*) → None

```
# SCPI: IPM:TYPE
driver.ipm.set_type_py(type_py = enums.IpmType.BINomial)
```

Sets the shape of the profile.

param type_py
STEPs| WAVeform| RLISt| LIST| SHAPe| RANDom| EQUation| PLUGin| RSTep| BINomial

set_unit(*unit: Units*) → None

```
# SCPI: IPM:UNIT
driver.ipm.set_unit(unit = enums.Units.DB)
```

Sets the units of the profile.

param unit
NONE| SEConds| HERTz| DB| DEGRees| PERCent

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ipm.clone()
```

Subgroups

5.13.1 Binomial

SCPI Commands :

```
IPM:BINomial:PVAL1
IPM:BINomial:VAL1
IPM:BINomial:VAL2
```

class BinomialCls

Binomial commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_pval_1() → float

```
# SCPI: IPM:BINomial:PVAL1
value: float = driver.ipm.binomial.get_pval_1()
```

Sets the probability of occurrence of value 1 in the binomial distribution function.

return
pval_1: float Range: 0 to 100, Unit: PCT

get_val_1() → float

```
# SCPI: IPM:BINomial:VAL1
value: float = driver.ipm.binomial.get_val_1()
```

Sets the values of the binomial distribution function.

return
val_1: No help available

get_val_2() → float

```
# SCPI: IPM:BINomial:VAL2
value: float = driver.ipm.binomial.get_val_2()
```

Sets the values of the binomial distribution function.

return
val_2: float Range: -1e+09 to 1e+09

set_pval_1(*pval_1: float*) → None

```
# SCPI: IPM:BINomial:PVAL1
driver.ipm.binomial.set_pval_1(pval_1 = 1.0)
```

Sets the probability of occurrence of value 1 in the binomial distribution function.

param pval_1
float Range: 0 to 100, Unit: PCT

set_val_1(*val_1: float*) → None

```
# SCPI: IPM:BINomial:VAL1
driver.ipm.binomial.set_val_1(val_1 = 1.0)
```

Sets the values of the binomial distribution function.

param val_1
float Range: -1e+09 to 1e+09

set_val_2(*val_2: float*) → None

```
# SCPI: IPM:BINomial:VAL2
driver.ipm.binomial.set_val_2(val_2 = 1.0)
```

Sets the values of the binomial distribution function.

param val_2
float Range: -1e+09 to 1e+09

5.13.2 ListPy

SCPI Commands :

```
IPM:LIST:BASE
IPM:LIST:CLEar
IPM:LIST:LOAD
IPM:LIST:SAVE
```

class ListPyCls

ListPy commands group definition. 13 total commands, 2 Subgroups, 4 group commands

clear() → None

```
# SCPI: IPM:LIST:CLEar
driver.ipm.listPy.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IPM:LIST:CLEar
driver.ipm.listPy.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_base() → BaseDomain

```
# SCPI: IPM:LIST:BASE
value: enums.BaseDomain = driver.ipm.listPy.get_base()
```

Sets the IPM profile base and defines how the steps repetition is defined.

return
base: PULSe| TIME PULSe Steps are repeated several times, as set with the command method RsPulseSeq.Ipm.ListPy.Item.repetition. TIME Steps are repeated for the defined time duration, as set with the command method RsPulseSeq.Ipm.ListPy.Item.time.

get_load() → str

```
# SCPI: IPM:LIST:LOAD
value: str = driver.ipm.listPy.get_load()
```

Loads an IPM profile form an ASCII file.

return
load: string File path, file name, and file extension

get_save() → str

```
# SCPI: IPM:LIST:SAVE
value: str = driver.ipm.listPy.get_save()
```

Stores the IPM profile as a file.

return
save: string File path incl. file name and extension.

set_base(*base: BaseDomain*) → None

```
# SCPI: IPM:LIST:BASE
driver.ipm.listPy.set_base(base = enums.BaseDomain.PULSe)
```

Sets the IPM profile base and defines how the steps repetition is defined.

param base
PULSe| TIME PULSe Steps are repeated several times, as set with the command method RsPulseSeq.Ipm.ListPy.Item.repetition. TIME Steps are repeated for the defined time duration, as set with the command method RsPulseSeq.Ipm.ListPy.Item.time.

set_load(*load: str*) → None

```
# SCPI: IPM:LIST:LOAD
driver.ipm.listPy.set_load(load = 'abc')
```

Loads an IPM profile form an ASCII file.

param load
string File path, file name, and file extension

set_save(*save: str*) → None

```
# SCPI: IPM:LIST:SAVE
driver.ipm.listPy.set_save(save = 'abc')
```

Stores the IPM profile as a file.

param save
string File path incl. file name and extension.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ipm.listPy.clone()
```

Subgroups

5.13.2.1 Firing

SCPI Commands :

```
IPM:LIST:FIRing:ENABle
IPM:LIST:FIRing:SEQuence
```

class FiringCls

Firing commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: IPM:LIST:FIRing:ENABle
value: bool = driver.ipm.listPy.firing.get_enable()
```

Enables using firing order for list-based IPM profiles.

return
enable: ON| OFF| 1| 0

get_sequence() → str

```
# SCPI: IPM:LIST:FIRing:SEQuence
value: str = driver.ipm.listPy.firing.get_sequence()
```

Sets the firing order sequence.

return
sequence: string

set_enable(*enable: bool*) → None

```
# SCPI: IPM:LIST:FIRing:ENABle
driver.ipm.listPy.firing.set_enable(enable = False)
```

Enables using firing order for list-based IPM profiles.

param enable
ON| OFF| 1| 0

set_sequence(*sequence: str*) → None

```
# SCPI: IPM:LIST:FIRing:SEQuence
driver.ipm.listPy.firing.set_sequence(sequence = 'abc')
```

Sets the firing order sequence.

param sequence
string

5.13.2.2 Item

SCPI Commands :

```
IPM:LIST:ITEM:COUNt
IPM:LIST:ITEM:DELete
IPM:LIST:ITEM:REPetition
IPM:LIST:ITEM:SELect
IPM:LIST:ITEM:TIME
IPM:LIST:ITEM:VALue
```

class ItemCls

Item commands group definition. 7 total commands, 1 Subgroups, 6 group commands

delete(*delete: float*) → None

```
# SCPI: IPM:LIST:ITEM:DELete
driver.ipm.listPy.item.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: IPM:LIST:ITEM:COUNt
value: float = driver.ipm.listPy.item.get_count()
```

Queries the number of existing items.

return
count: integer

get_repetition() → float

```
# SCPI: IPM:LIST:ITEM:REPetition
value: float = driver.ipm.listPy.item.get_repetition()
```

Sets the number of times a list item is repeated.

return
repetition: float Range: 1 to 1e+09

get_select() → float

```
# SCPI: IPM:LIST:ITEM:SELect
value: float = driver.ipm.listPy.item.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_time() → float

```
# SCPI: IPM:LIST:ITEM:TIME
value: float = driver.ipm.listPy.item.get_time()
```

Sets how long a list item is repeated.

return
time: float Range: 0 to 1e+09

get_value() → float

```
# SCPI: IPM:LIST:ITEM:VALue
value: float = driver.ipm.listPy.item.get_value()
```

Sets the value of the selected list item.

return
value: float Range: -1e+11 to 1e+11

set_repetition(*repetition: float*) → None

```
# SCPI: IPM:LIST:ITEM:REPetition
driver.ipm.listPy.item.set_repetition(repetition = 1.0)
```

Sets the number of times a list item is repeated.

param repetition
float Range: 1 to 1e+09

set_select(*select: float*) → None

```
# SCPI: IPM:LIST:ITEM:SELect
driver.ipm.listPy.item.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_time(*time: float*) → None

```
# SCPI: IPM:LIST:ITEM:TIME
driver.ipm.listPy.item.set_time(time = 1.0)
```

Sets how long a list item is repeated.

param time
float Range: 0 to 1e+09

set_value(*value: float*) → None

```
# SCPI: IPM:LIST:ITEM:VALue
driver.ipm.listPy.item.set_value(value = 1.0)
```

Sets the value of the selected list item.

param value
float Range: -1e+11 to 1e+11

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ipm.listPy.item.clone()
```

Subgroups

5.13.2.2.1 Add

SCPI Command :

```
IPM:LIST:ITEM:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: IPM:LIST:ITEM:ADD
driver.ipm.listPy.item.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IPM:LIST:ITEM:ADD
driver.ipm.listPy.item.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.13.3 Plugin

SCPI Command :

```
IPM:PLUGin:NAME
```

class PluginCls

Plugin commands group definition. 6 total commands, 1 Subgroups, 1 group commands

get_name() → str

```
# SCPI: IPM:PLUGin:NAME
value: str = driver.ipm.plugin.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: IPM:PLUGin:NAME
driver.ipm.plugin.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ipm.plugin.clone()
```

Subgroups

5.13.3.1 Variable

SCPI Commands :

```
IPM:PLUGin:VARiable:CATalog
IPM:PLUGin:VARiable:RESet
IPM:PLUGin:VARiable:VALue
```

class VariableCls

Variable commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_catalog() → str

```
# SCPI: IPM:PLUGin:VARiable:CATalog
value: str = driver.ipm.plugin.variable.get_catalog()
```

Queries the variables used in the plugin.

return
catalog: string

get_value() → str

```
# SCPI: IPM:PLUGin:VARiable:VALue
value: str = driver.ipm.plugin.variable.get_value()
```

Sets the values of the selected variable.

return
value: string

reset() → None

```
# SCPI: IPM:PLUGin:VARiable:RESet
driver.ipm.plugin.variable.reset()
```

No command help available

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: IPM:PLUGin:VARiable:RESet
driver.ipm.plugin.variable.reset_with_opc()
```

No command help available

Same as reset, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*value: str*) → None

```
# SCPI: IPM:PLUGin:VARiable:VALue
driver.ipm.plugin.variable.set_value(value = 'abc')
```

Sets the values of the selected variable.

param value
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ipm.plugin.variable.clone()
```

Subgroups

5.13.3.1.1 Select

SCPI Commands :

```
IPM:PLUGin:VARiable:SELect:ID
IPM:PLUGin:VARiable:SELect
```

class SelectCls

Select commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_id() → float

```
# SCPI: IPM:PLUGin:VARiable:SELect:ID
value: float = driver.ipm.plugin.variable.select.get_id()
```

No command help available

return
idn: No help available

get_value() → str

```
# SCPI: IPM:PLUGin:VARiable:SELect
value: str = driver.ipm.plugin.variable.select.get_value()
```

Selects a plugin variable.

return
select: string

set_id(*idn: float*) → None

```
# SCPI: IPM:PLUGin:VARiable:SELect:ID
driver.ipm.plugin.variable.select.set_id(idn = 1.0)
```

No command help available

param idn
No help available

set_value(*select: str*) → None

```
# SCPI: IPM:PLUGin:VARiable:SELect
driver.ipm.plugin.variable.select.set_value(select = 'abc')
```

Selects a plugin variable.

param select
string

5.13.4 Random

SCPI Command :

```
IPM:RANDom:DISTribution
```

class RandomCls

Random commands group definition. 9 total commands, 3 Subgroups, 1 group commands

get_distribution() → RandomDistribution

```
# SCPI: IPM:RANDom:DISTribution
value: enums.RandomDistribution = driver.ipm.random.get_distribution()
```

Sets the distribution function.

return
distribution: UNIForm| NORMal| U

set_distribution(*distribution: RandomDistribution*) → None

```
# SCPI: IPM:RANDom:DISTribution
driver.ipm.random.set_distribution(distribution = enums.RandomDistribution.
↪NORMal)
```

Sets the distribution function.

param distribution
UNIForm| NORMal| U

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ipm.random.clone()
```

Subgroups

5.13.4.1 Normal

SCPI Commands :

```
IPM:RANDom:NORMal:LIMit
IPM:RANDom:NORMal:MEAN
IPM:RANDom:NORMal:STD
```

class NormalCls

Normal commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_limit() → float

```
# SCPI: IPM:RANDom:NORMal:LIMit
value: float = driver.ipm.random.normal.get_limit()
```

Sets the limit parameter of the normal distribution function.

return
limit: float Range: -1e+09 to 1e+09

get_mean() → float

```
# SCPI: IPM:RANDom:NORMal:MEAN
value: float = driver.ipm.random.normal.get_mean()
```

Sets the mean parameter of the normal distribution function.

return
mean: float Range: -1e+09 to 1e+09

get_std() → float

```
# SCPI: IPM:RANDom:NORMal:STD
value: float = driver.ipm.random.normal.get_std()
```

Sets the standard deviation parameter of the normal distribution function.

return
std: float Range: 1e-09 to 1e+06

set_limit(*limit: float*) → None

```
# SCPI: IPM:RANDom:NORMal:LIMit
driver.ipm.random.normal.set_limit(limit = 1.0)
```

Sets the limit parameter of the normal distribution function.

param limit
float Range: -1e+09 to 1e+09

set_mean(*mean: float*) → None

```
# SCPI: IPM:RANDom:NORMal:MEAN
driver.ipm.random.normal.set_mean(mean = 1.0)
```

Sets the mean parameter of the normal distribution function.

param mean
float Range: -1e+09 to 1e+09

set_std(*std: float*) → None

```
# SCPI: IPM:RANDom:NORMal:STD
driver.ipm.random.normal.set_std(std = 1.0)
```

Sets the standard deviation parameter of the normal distribution function.

param std
float Range: 1e-09 to 1e+06

5.13.4.2 U

SCPI Commands :

```
IPM:RANDom:U:CENTer
IPM:RANDom:U:RANGe
```

class UCls

U commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_center() → float

```
# SCPI: IPM:RANDom:U:CENTer
value: float = driver.ipm.random.u.get_center()
```

Sets the center parameter of the U distribution.

return
center: float Range: -1e+09 to 1e+09

get_range() → float

```
# SCPI: IPM:RANDom:U:RANGe
value: float = driver.ipm.random.u.get_range()
```

Sets the range parameter of the U distribution.

return
range_py: float Range: 1e-09 to 1e+09

set_center(*center: float*) → None

```
# SCPI: IPM:RANDom:U:CENTer
driver.ipm.random.u.set_center(center = 1.0)
```

Sets the center parameter of the U distribution.

param center
float Range: -1e+09 to 1e+09

set_range(*range_py: float*) → None

```
# SCPI: IPM:RANDom:U:RANGe
driver.ipm.random.u.set_range(range_py = 1.0)
```

Sets the range parameter of the U distribution.

param range_py
float Range: 1e-09 to 1e+09

5.13.4.3 Uniform

SCPI Commands :

```
IPM:RANDom:UNIForm:MAXimum
IPM:RANDom:UNIForm:MINimum
IPM:RANDom:UNIForm:STEP
```

class UniformCls

Uniform commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_maximum() → float

```
# SCPI: IPM:RANDom:UNIForm:MAXimum
value: float = driver.ipm.random.uniform.get_maximum()
```

Sets the range of the uniform distribution function.

return
maximum: float Range: -1e+09 to 1e+09

get_minimum() → float

```
# SCPI: IPM:RANDom:UNIForm:MINimum
value: float = driver.ipm.random.uniform.get_minimum()
```

Sets the range of the uniform distribution function.

return
minimum: No help available

get_step() → float

```
# SCPI: IPM:RANDom:UNIForm:STEP
value: float = driver.ipm.random.uniform.get_step()
```

Sets the granularity of the uniform distribution function.

return
step: float Range: 1e-09 to 1e+09

set_maximum(*maximum: float*) → None

```
# SCPI: IPM:RANDom:UNIForm:MAXimum
driver.ipm.random.uniform.set_maximum(maximum = 1.0)
```

Sets the range of the uniform distribution function.

param maximum
float Range: -1e+09 to 1e+09

set_minimum(*minimum: float*) → None

```
# SCPI: IPM:RANDom:UNIForm:MINimum
driver.ipm.random.uniform.set_minimum(minimum = 1.0)
```

Sets the range of the uniform distribution function.

param minimum
float Range: -1e+09 to 1e+09

`set_step`(*step: float*) → None

```
# SCPI: IPM:RANDom:UNIForm:STEP
driver.ipm.random.uniform.set_step(step = 1.0)
```

Sets the granularity of the uniform distribution function.

param step
float Range: 1e-09 to 1e+09

5.13.5 Rlist

SCPI Commands :

```
IPM:RLISt:BASE
IPM:RLISt:BURSt
IPM:RLISt:PERiod
IPM:RLISt:REUSe
```

`class RlistCls`

Rlist commands group definition. 4 total commands, 0 Subgroups, 4 group commands

`get_base`() → BaseDomainB

```
# SCPI: IPM:RLISt:BASE
value: enums.BaseDomainB = driver.ipm.rlist.get_base()
```

Sets the IPM profile base and defines how the increments repetition is defined.

return
base: LENGth| TIME LENGth Increments are repeated several times, as set with the command method RsPulseSeq.Ipm.Rlist.burst. TIME Increments are repeated for the defined time duration, as set with the command method RsPulseSeq.Ipm.Rlist.period.

`get_burst`() → float

```
# SCPI: IPM:RLISt:BURSt
value: float = driver.ipm.rlist.get_burst()
```

Defines how many times an increment is repeated.

return
burst: float Range: 1 to 8192

`get_period`() → float

```
# SCPI: IPM:RLISt:PERiod
value: float = driver.ipm.rlist.get_period()
```

Sets how long an increment is repeated.

return
period: float Range: 1e-09 to 1e+09

get_reuse() → bool

```
# SCPI: IPM:RLISt:REUSe
value: bool = driver.ipm.rlist.get_reuse()
```

If disabled, each value is used only once.

return
reuse: ON| OFF| 1| 0

set_base(*base: BaseDomainB*) → None

```
# SCPI: IPM:RLISt:BASE
driver.ipm.rlist.set_base(base = enums.BaseDomainB.LENGth)
```

Sets the IPM profile base and defines how the increments repetition is defined.

param base
LENGth| TIME LENGth Increments are repeated several times, as set with the command method RsPulseSeq.Ipm.Rlist.burst. TIME Increments are repeated for the defined time duration, as set with the command method RsPulseSeq.Ipm.Rlist.period.

set_burst(*burst: float*) → None

```
# SCPI: IPM:RLISt:BURSt
driver.ipm.rlist.set_burst(burst = 1.0)
```

Defines how many times an increment is repeated.

param burst
float Range: 1 to 8192

set_period(*period: float*) → None

```
# SCPI: IPM:RLISt:PERiod
driver.ipm.rlist.set_period(period = 1.0)
```

Sets how long an increment is repeated.

param period
float Range: 1e-09 to 1e+09

set_reuse(*reuse: bool*) → None

```
# SCPI: IPM:RLISt:REUSe
driver.ipm.rlist.set_reuse(reuse = False)
```

If disabled, each value is used only once.

param reuse
ON| OFF| 1| 0

5.13.6 Rstep

SCPI Commands :

```
IPM:RSTep:MAXimum
IPM:RSTep:MINimum
IPM:RSTep:PERiod
```

class RstepCls

Rstep commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_maximum() → float

```
# SCPI: IPM:RSTep:MAXimum
value: float = driver.ipm.rstep.get_maximum()
```

Sets the value range.

return
maximum: float Range: 0 to 1e+11

get_minimum() → float

```
# SCPI: IPM:RSTep:MINimum
value: float = driver.ipm.rstep.get_minimum()
```

Sets the value range.

return
minimum: No help available

get_period() → float

```
# SCPI: IPM:RSTep:PERiod
value: float = driver.ipm.rstep.get_period()
```

Sets the pattern length.

return
period: float Range: 0 to 4096

set_maximum(*maximum: float*) → None

```
# SCPI: IPM:RSTep:MAXimum
driver.ipm.rstep.set_maximum(maximum = 1.0)
```

Sets the value range.

param maximum
float Range: 0 to 1e+11

set_minimum(*minimum: float*) → None

```
# SCPI: IPM:RSTep:MINimum
driver.ipm.rstep.set_minimum(minimum = 1.0)
```

Sets the value range.

param minimum
float Range: 0 to 1e+11

set_period(*period: float*) → None

```
# SCPI: IPM:RSTep:PERiod
driver.ipm.rstep.set_period(period = 1.0)
```

Sets the pattern length.

param period
float Range: 0 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ipm.rstep.clone()
```

Subgroups

5.13.6.1 Step

SCPI Commands :

```
IPM:RSTep:STEP:MAXimum
IPM:RSTep:STEP:MINimum
```

class StepCls

Step commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_maximum() → float

```
# SCPI: IPM:RSTep:STEP:MAXimum
value: float = driver.ipm.rstep.step.get_maximum()
```

Sets the step size range.

return
maximum: float Range: 0.1 to 0.5

get_minimum() → float

```
# SCPI: IPM:RSTep:STEP:MINimum
value: float = driver.ipm.rstep.step.get_minimum()
```

Sets the step size range.

return
minimum: No help available

set_maximum(*maximum: float*) → None

```
# SCPI: IPM:RSTep:STEP:MAXimum
driver.ipm.rstep.step.set_maximum(maximum = 1.0)
```

Sets the step size range.

param maximum
float Range: 0.1 to 0.5

set_minimum(*minimum: float*) → None

```
# SCPI: IPM:RSTep:STEP:MINimum
driver.ipm.rstep.step.set_minimum(minimum = 1.0)
```

Sets the step size range.

param minimum
float Range: 0.1 to 0.5

5.13.7 Shape

SCPI Commands :

```
IPM:SHAPe:BASE
IPM:SHAPe:COUNt
IPM:SHAPe:INTerpol
IPM:SHAPe:PERiod
```

class ShapeCls

Shape commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_base() → BaseDomain

```
# SCPI: IPM:SHAPe:BASE
value: enums.BaseDomain = driver.ipm.shape.get_base()
```

Defines the way the list items are processed.

return
base: PULSe| TIME

get_count() → float

```
# SCPI: IPM:SHAPe:COUNt
value: float = driver.ipm.shape.get_count()
```

Sets the number of pulses for that the data from the list is used.

return
count: integer Range: 1 to 1e+09

get_interpol() → Interpolation

```
# SCPI: IPM:SHAPe:INTerpol
value: enums.Interpolation = driver.ipm.shape.get_interpol()
```

Enables a linear transition between the increments.

return
interpol: LINear| NONE

get_period() → float

```
# SCPI: IPM:SHAPe:PERiod
value: float = driver.ipm.shape.get_period()
```

Sets the period of time over that the list items are equally distributed.

return
period: float Range: 1e-09 to 1e+09

set_base(*base: BaseDomain*) → None

```
# SCPI: IPM:SHAPe:BASE
driver.ipm.shape.set_base(base = enums.BaseDomain.PULSe)
```

Defines the way the list items are processed.

param base
PULSe| TIME

set_count(*count: float*) → None

```
# SCPI: IPM:SHAPe:COUNt
driver.ipm.shape.set_count(count = 1.0)
```

Sets the number of pulses for that the data from the list is used.

param count
integer Range: 1 to 1e+09

set_interpol(*interpol: Interpolation*) → None

```
# SCPI: IPM:SHAPe:INTerpol
driver.ipm.shape.set_interpol(interpol = enums.Interpolation.LINear)
```

Enables a linear transition between the increments.

param interpol
LINear| NONE

set_period(*period: float*) → None

```
# SCPI: IPM:SHAPe:PERiod
driver.ipm.shape.set_period(period = 1.0)
```

Sets the period of time over that the list items are equally distributed.

param period
float Range: 1e-09 to 1e+09

5.13.8 Step

SCPI Commands :

```
IPM:STEP:BASE
IPM:STEP:BURSt
IPM:STEP:INCRement
IPM:STEP:PERiod
IPM:STEP:STARt
IPM:STEP:STEPs
```

class StepCls

Step commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_base() → BaseDomainB

```
# SCPI: IPM:STEP:BASE
value: enums.BaseDomainB = driver.ipm.step.get_base()
```

Sets the IPM profile base and defines how the increments repetition is defined.

return
base: LENGth| TIME LENGth Steps are repeated several times, as set with the command method RsPulseSeq.Ipm.Step.burst. TIME Steps are repeated for the defined time duration, as set with the command method RsPulseSeq.Ipm.Step.period.

get_burst() → float

```
# SCPI: IPM:STEP:BURSt
value: float = driver.ipm.step.get_burst()
```

Sets the number of times an increment is repeated.

return
burst: float Range: 1 to 1000

get_increment() → float

```
# SCPI: IPM:STEP:INCRement
value: float = driver.ipm.step.get_increment()
```

Sets the step size.

return
increment: float Range: -1e+09 to 1e+09

get_period() → float

```
# SCPI: IPM:STEP:PERiod
value: float = driver.ipm.step.get_period()
```

Sets how long an increment is repeated.

return
period: float Range: 1e-09 to 1e+09

get_start() → float

```
# SCPI: IPM:STEP:STARt
value: float = driver.ipm.step.get_start()
```

Sets the start value.

return
start: float Range: -1e+09 to 1e+09

get_steps() → float

```
# SCPI: IPM:STEP:STEPs
value: float = driver.ipm.step.get_steps()
```

Sets the number of steps.

return
steps: float Range: 1 to 10000

set_base(*base: BaseDomainB*) → None

```
# SCPI: IPM:STEP:BASE
driver.ipm.step.set_base(base = enums.BaseDomainB.LENGth)
```

Sets the IPM profile base and defines how the increments repetition is defined.

param base
LENGth| TIME LENGth Steps are repeated several times, as set with the command method RsPulseSeq.Ipm.Step.burst. TIME Steps are repeated for the defined time duration, as set with the command method RsPulseSeq.Ipm.Step.period.

set_burst(*burst: float*) → None

```
# SCPI: IPM:STEP:BURSt
driver.ipm.step.set_burst(burst = 1.0)
```

Sets the number of times an increment is repeated.

param burst
float Range: 1 to 1000

set_increment(*increment: float*) → None

```
# SCPI: IPM:STEP:INCRement
driver.ipm.step.set_increment(increment = 1.0)
```

Sets the step size.

param increment
float Range: -1e+09 to 1e+09

set_period(*period: float*) → None

```
# SCPI: IPM:STEP:PERiod
driver.ipm.step.set_period(period = 1.0)
```

Sets how long an increment is repeated.

param period
float Range: 1e-09 to 1e+09

`set_start`(*start: float*) → None

```
# SCPI: IPM:STEP:STARt
driver.ipm.step.set_start(start = 1.0)
```

Sets the start value.

param start
float Range: -1e+09 to 1e+09

`set_steps`(*steps: float*) → None

```
# SCPI: IPM:STEP:STEPs
driver.ipm.step.set_steps(steps = 1.0)
```

Sets the number of steps.

param steps
float Range: 1 to 10000

5.13.9 Waveform

SCPI Commands :

```
IPM:WAVeform:BASE
IPM:WAVeform:COUNt
IPM:WAVeform:OFFSet
IPM:WAVeform:PERiod
IPM:WAVeform:PHASe
IPM:WAVeform:PKPK
IPM:WAVeform:TYPE
```

`class WaveformCls`

Waveform commands group definition. 7 total commands, 0 Subgroups, 7 group commands

`get_base`() → BaseDomain

```
# SCPI: IPM:WAVeform:BASE
value: enums.BaseDomain = driver.ipm.waveform.get_base()
```

Defines how the waveform period is defined, as a time duration or as a number of pulses.

return
base: PULSe| TIME

`get_count`() → float

```
# SCPI: IPM:WAVeform:COUNt
value: float = driver.ipm.waveform.get_count()
```

Sets the waveform period as number of pulses.

return
count: integer Range: 1 to 1e+09

get_offset() → float

```
# SCPI: IPM:WAVeform:OFFSet
value: float = driver.ipm.waveform.get_offset()
```

Shifts the profile by the selected offset.

return
offset: float Range: -1e+09 to 1e+09

get_period() → float

```
# SCPI: IPM:WAVeform:PERiod
value: float = driver.ipm.waveform.get_period()
```

Sets the waveform period.

return
period: float Range: 1e-09 to 1e+09, Unit: sec

get_phase() → float

```
# SCPI: IPM:WAVeform:PHASe
value: float = driver.ipm.waveform.get_phase()
```

Enables a phase offset to change the start phase of the sine wave.

return
phase: float Range: -1e+09 to 1e+09, Unit: sec

get_pkpk() → float

```
# SCPI: IPM:WAVeform:PKPK
value: float = driver.ipm.waveform.get_pkpk()
```

Sets the value range of the linear ramp profile or the period of the sine profile.

return
pkpk: float Range: 1e-09 to 1e+09, Unit: sec

get_type_py() → WaveformShape

```
# SCPI: IPM:WAVeform:TYPE
value: enums.WaveformShape = driver.ipm.waveform.get_type_py()
```

Sets the profile shape.

return
type_py: RAMP| SINE| TRIangular

set_base(*base: BaseDomain*) → None

```
# SCPI: IPM:WAVeform:BASE
driver.ipm.waveform.set_base(base = enums.BaseDomain.PULSe)
```

Defines how the waveform period is defined, as a time duration or as a number of pulses.

param base
PULSe| TIME

set_count(*count: float*) → None

```
# SCPI: IPM:WAVeform:COUNt
driver.ipm.waveform.set_count(count = 1.0)
```

Sets the waveform period as number of pulses.

param count
integer Range: 1 to 1e+09

set_offset(*offset: float*) → None

```
# SCPI: IPM:WAVeform:OFFSet
driver.ipm.waveform.set_offset(offset = 1.0)
```

Shifts the profile by the selected offset.

param offset
float Range: -1e+09 to 1e+09

set_period(*period: float*) → None

```
# SCPI: IPM:WAVeform:PERiod
driver.ipm.waveform.set_period(period = 1.0)
```

Sets the waveform period.

param period
float Range: 1e-09 to 1e+09, Unit: sec

set_phase(*phase: float*) → None

```
# SCPI: IPM:WAVeform:PHASe
driver.ipm.waveform.set_phase(phase = 1.0)
```

Enables a phase offset to change the start phase of the sine wave.

param phase
float Range: -1e+09 to 1e+09, Unit: sec

set_pkpk(*pkpk: float*) → None

```
# SCPI: IPM:WAVeform:PKPK
driver.ipm.waveform.set_pkpk(pkpk = 1.0)
```

Sets the value range of the linear ramp profile or the period of the sine profile.

param pkpk
float Range: 1e-09 to 1e+09, Unit: sec

set_type_py(*type_py: WaveformShape*) → None

```
# SCPI: IPM:WAVeform:TYPE
driver.ipm.waveform.set_type_py(type_py = enums.WaveformShape.RAMP)
```

Sets the profile shape.

param type_py
RAMP| SINE| TRIangular

5.14 Lserver

SCPI Commands :

```
LSERver:OPTions
LSERver:READy
LSERver:STATus
```

class LserverCls

Lserver commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_options() → str

```
# SCPI: LSERver:OPTions
value: str = driver.lserver.get_options()
```

Queries the available options.

return
options: string

get_ready() → bool

```
# SCPI: LSERver:READy
value: bool = driver.lserver.get_ready()
```

Queries the status of the license server.

return
ready: ON| OFF| 1| 0

get_status() → str

```
# SCPI: LSERver:STATus
value: str = driver.lserver.get_status()
```

Queries the status of the license server.

return
status: string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.lserver.clone()
```

Subgroups

5.14.1 Apply

SCPI Command :

```
LSERver:APPLy
```

class ApplyCls

Apply commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: LSERver:APPLy
driver.lserver.apply.set()
```

Applies the changes.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: LSERver:APPLy
driver.lserver.apply.set_with_opc()
```

Applies the changes.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.15 MsgLog

SCPI Commands :

```
MSGLog:ERRor
MSGLog:POPup
```

class MsgLogCls

MsgLog commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_error() → str

```
# SCPI: MSGLog:ERRor
value: str = driver.msgLog.get_error()
```

Queries the last error listed in the 'Message Log' dialog.

return
error: string

get_popup() → bool

```
# SCPI: MSGLog:POPup
value: bool = driver.msgLog.get_popup()
```

Opens/closes the 'Message Log' dialog.

return
popup: ON| OFF| 1| 0

set_popup(*popup: bool*) → None

```
# SCPI: MSGLog:POPup
driver.msgLog.set_popup(popup = False)
```

Opens/closes the 'Message Log' dialog.

param popup
ON| OFF| 1| 0

5.16 Platform

SCPI Commands :

```
PLATform:ID
PLATform:CATalog
PLATform:COMMent
PLATform:CREate
PLATform:NAME
PLATform:REMove
PLATform:SELect
```

class PlatformCls

Platform commands group definition. 31 total commands, 1 Subgroups, 7 group commands

get_catalog() → str

```
# SCPI: PLATform:CATalog
value: str = driver.platform.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: PLATform:COMMent
value: str = driver.platform.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_id() → float

```
# SCPI: PLATform:ID
value: float = driver.platform.get_id()
```

Platform identifier.

return
idn: float Range: 1 to 65536

get_name() → str

```
# SCPI: PLATform:NAME
value: str = driver.platform.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: PLATform:SELect
value: str = driver.platform.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the . . . :CREate or . . . :NAME command. To query the existing elements, use the . . . :CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_comment(*comment: str*) → None

```
# SCPI: PLATform:COMMent
driver.platform.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: PLATform:CREate
driver.platform.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_id(*idn: float*) → None

```
# SCPI: PLATform:ID
driver.platform.set_id(idn = 1.0)
```

Platform identifier.

param idn
float Range: 1 to 65536

set_name(*name: str*) → None

```
# SCPI: PLATform:NAME
driver.platform.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: PLATform:REMove
driver.platform.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: PLATform:SELect
driver.platform.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.platform.clone()
```

Subgroups

5.16.1 Emitter

SCPI Commands :

```
PLATform:EMITter:ALIas
PLATform:EMITter:ANGLe
PLATform:EMITter:AZIMuth
PLATform:EMITter:BM
```

(continues on next page)

(continued from previous page)

```
PLATform:EMITter:BMID
PLATform:EMITter:CLEar
PLATform:EMITter:DELete
PLATform:EMITter:ELEVation
PLATform:EMITter:HEIGht
PLATform:EMITter:RADius
PLATform:EMITter:ROLL
PLATform:EMITter:SELect
PLATform:EMITter:X
PLATform:EMITter:Y
PLATform:EMITter
```

class EmitterCls

Emitter commands group definition. 24 total commands, 3 Subgroups, 15 group commands

clear() → None

```
# SCPI: PLATform:EMITter:CLEar
driver.platform.emitter.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PLATform:EMITter:CLEar
driver.platform.emitter.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PLATform:EMITter:DELete
driver.platform.emitter.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: PLATform:EMITter:ALIas
value: str = driver.platform.emitter.get_alias()
```

Sets an alias name for the selected platform emitter element.

return
alias: string

get_angle() → float

```
# SCPI: PLATform:EMITter:ANGLe
value: float = driver.platform.emitter.get_angle()
```

You can set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.radius.

- method RsPulseSeq.Platform.Emitter.angle sets the angle of the emitter element on the azimuth plane, relative to the platform's heading.
- method RsPulseSeq.Platform.Emitter.radius sets the distance of the emitter element on the azimuth plane, relative to the platform's origin.

return
angle: float Range: 0 to 360

get_azimuth() → float

```
# SCPI: PLATform:EMITter:AZIMuth
value: float = driver.platform.emitter.get_azimuth()
```

Angle of the emitter element's pointing direction relative to the platform's heading.

return
azimuth: float Range: 0 to 360

get_bm() → str

```
# SCPI: PLATform:EMITter:BM
value: str = driver.platform.emitter.get_bm()
```

No command help available

return
bm: No help available

get_bmid() → str

```
# SCPI: PLATform:EMITter:BMID
value: str = driver.platform.emitter.get_bmid()
```

No command help available

return
bmid: No help available

get_elevation() → float

```
# SCPI: PLATform:EMITter:ELEVation
value: float = driver.platform.emitter.get_elevation()
```

Elevation of the emitter item's pointing direction, relative to the azimuth plane.

return
elevation: float Range: -90 to 90

get_height() → float

```
# SCPI: PLATform:EMITter:HEIGht
value: float = driver.platform.emitter.get_height()
```

Height of the selected emitter element relative to the platform's origin. Can be used, for example, to differentiate between:

- Radars mounted on different parts of a ship or aircraft.
- Various radars situated across a land-based radar installation.

return
height: float Range: -500 to 500

get_radius() → float

```
# SCPI: PLATform:EMITter:RADius
value: float = driver.platform.emitter.get_radius()
```

You can set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.angle.

- method RsPulseSeq.Platform.Emitter.angle sets the angle of the emitter element on the azimuth plane, relative to the platform's heading.
- method RsPulseSeq.Platform.Emitter.radius sets the distance of the emitter element on the azimuth plane, relative to the platform's origin.

return
radius: float Range: 0 to 2000

get_roll() → float

```
# SCPI: PLATform:EMITter:ROLL
value: float = driver.platform.emitter.get_roll()
```

Roll of the emitter item's pointing direction relative to the platform's up direction. Can be used, for example, to simulate the emissions from a mast-mounted radar on a marine platform affected by wind.

return
roll: float Range: -180 to 180

get_select() → float

```
# SCPI: PLATform:EMITter:SELect
value: float = driver.platform.emitter.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the . . . :CREate or . . . :NAME command. To query the existing elements, use the . . . :CATalog? command. For example, method RsPulseSeq.Repository.catalog.

get_value() → str

```
# SCPI: PLATform:EMITter
value: str = driver.platform.emitter.get_value()
```

The string must be unique within the repository. Letters, numbers, spaces and some special characters can be used.
INTRO_CMD_HELP: Examples of special characters:

- Supported: !$% =?-+_.
- Not supported: &/:{umlaut}{umlaut}{umlaut}

return
emitter: string

get_x() → float

```
# SCPI: PLATform:EMITter:X
value: float = driver.platform.emitter.get_x()
```

Set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.y.

INTRO_CMD_HELP: X and Y represent the two principle axis of the platform.

- The Y-axis represents the axis along the center-line of the platform. This axis:

Table Header:

- Corresponds to its heading.
- Passes through the origin.
- The X-axis:

Table Header:

- Is at right-angles to the Y-axis.
- Passes through the origin.
- method RsPulseSeq.Platform.Emitter.y sets the distance of the emitter element from the origin, along the Y-axis. Positive values are towards the heading.
- method RsPulseSeq.Platform.Emitter.x sets the distance of the emitter element from the origin, along the X-axis.

return
x: float Range: -2000 to 2000

get_y() → float

```
# SCPI: PLATform:EMITter:Y
value: float = driver.platform.emitter.get_y()
```

Set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.x.

INTRO_CMD_HELP: X and Y represent the two principle axis of the platform.

- The Y-axis represents the axis along the center-line of the platform. This axis:

Table Header:

- Corresponds to its heading.
- Passes through the origin.
- The X-axis:

Table Header:

- Is at right-angles to the Y-axis.
- Passes through the origin.

INTRO_CMD_HELP: X and Y represent the two principle axis of the platform.

- method RsPulseSeq.Platform.Emitter.y sets the distance of the emitter element from the origin, along the Y-axis. Positive values are towards the heading. Step = 0.01 m
- method RsPulseSeq.Platform.Emitter.x sets the distance of the emitter element from the origin, along the X-axis. Step = 0.01 m

return
y: float Range: -2000 to 2000

set_alias(*alias: str*) → None

```
# SCPI: PLATform:EMITter:ALIas
driver.platform.emitter.set_alias(alias = 'abc')
```

Sets an alias name for the selected platform emitter element.

param alias
string

set_angle(*angle: float*) → None

```
# SCPI: PLATform:EMITter:ANGLe
driver.platform.emitter.set_angle(angle = 1.0)
```

You can set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.radius.

- method RsPulseSeq.Platform.Emitter.angle sets the angle of the emitter element on the azimuth plane, relative to the platform's heading.
- method RsPulseSeq.Platform.Emitter.radius sets the distance of the emitter element on the azimuth plane, relative to the platform's origin.

param angle
float Range: 0 to 360

set_azimuth(*azimuth: float*) → None

```
# SCPI: PLATform:EMITter:AZIMuth
driver.platform.emitter.set_azimuth(azimuth = 1.0)
```

Angle of the emitter element's pointing direction relative to the platform's heading.

param azimuth
float Range: 0 to 360

set_bm(*bm: str*) → None

```
# SCPI: PLATform:EMITter:BM
driver.platform.emitter.set_bm(bm = 'abc')
```

No command help available

param bm
No help available

set_bmid(*bmid: str*) → None

```
# SCPI: PLATform:EMITter:BMID
driver.platform.emitter.set_bmid(bmid = 'abc')
```

No command help available

param bmid
No help available

set_elevation(*elevation: float*) → None

```
# SCPI: PLATform:EMITter:ELEVation
driver.platform.emitter.set_elevation(elevation = 1.0)
```

Elevation of the emitter item's pointing direction, relative to the azimuth plane.

param elevation
float Range: -90 to 90

set_height(*height: float*) → None

```
# SCPI: PLATform:EMITter:HEIGht
driver.platform.emitter.set_height(height = 1.0)
```

Height of the selected emitter element relative to the platform's origin. Can be used, for example, to differentiate between:

- Radars mounted on different parts of a ship or aircraft.
- Various radars situated across a land-based radar installation.

param height
float Range: -500 to 500

set_radius(*radius: float*) → None

```
# SCPI: PLATform:EMITter:RADius
driver.platform.emitter.set_radius(radius = 1.0)
```

You can set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.angle.

- method RsPulseSeq.Platform.Emitter.angle sets the angle of the emitter element on the azimuth plane, relative to the platform's heading.
- method RsPulseSeq.Platform.Emitter.radius sets the distance of the emitter element on the azimuth plane, relative to the platform's origin.

param radius
float Range: 0 to 2000

set_roll(*roll: float*) → None

```
# SCPI: PLATform:EMITter:ROLL
driver.platform.emitter.set_roll(roll = 1.0)
```

Roll of the emitter item's pointing direction relative to the platform's up direction. Can be used, for example, to simulate the emissions from a mast-mounted radar on a marine platform affected by wind.

param roll
float Range: -180 to 180

set_select(*select: float*) → None

```
# SCPI: PLATform:EMITter:SELect
driver.platform.emitter.set_select(select = 1.0)
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_value(*emitter: str*) → None

```
# SCPI: PLATform:EMITter
driver.platform.emitter.set_value(emitter = 'abc')
```

The string must be unique within the repository. Letters, numbers, spaces and some special characters can be used.
INTRO_CMD_HELP: Examples of special characters:

- Supported: !$% =?-+_.
- Not supported: &/:{umlaut}{umlaut}{umlaut}

param emitter
string

set_x(*x: float*) → None

```
# SCPI: PLATform:EMITter:X
driver.platform.emitter.set_x(x = 1.0)
```

Set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.y.

INTRO_CMD_HELP: X and Y represent the two principle axis of the platform.

- The Y-axis represents the axis along the center-line of the platform. This axis:

Table Header:

- Corresponds to its heading.
- Passes through the origin.
- The X-axis:

Table Header:

- Is at right-angles to the Y-axis.
- Passes through the origin.

INTRO_CMD_HELP: X and Y represent the two principle axis of the platform.

- method RsPulseSeq.Platform.Emitter.y sets the distance of the emitter element from the origin, along the Y-axis. Positive values are towards the heading.
- method RsPulseSeq.Platform.Emitter.x sets the distance of the emitter element from the origin, along the X-axis.

param x
float Range: -2000 to 2000

set_y(*y: float*) → None

```
# SCPI: PLATform:EMITter:Y
driver.platform.emitter.set_y(y = 1.0)
```

Set the position of the selected emitter relative to the platform's origin, using this command combined with method RsPulseSeq.Platform.Emitter.x.

INTRO_CMD_HELP: X and Y represent the two principle axis of the platform.

- The Y-axis represents the axis along the center-line of the platform. This axis:

Table Header:

- Corresponds to its heading.
- Passes through the origin.
- The X-axis:

Table Header:

- Is at right-angles to the Y-axis.
- Passes through the origin.

INTRO_CMD_HELP: X and Y represent the two principle axis of the platform.

- method RsPulseSeq.Platform.Emitter.y sets the distance of the emitter element from the origin, along the Y-axis. Positive values are towards the heading. Step = 0.01 m
- method RsPulseSeq.Platform.Emitter.x sets the distance of the emitter element from the origin, along the X-axis. Step = 0.01 m

param y
float Range: -2000 to 2000

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.platform.emitter.clone()
```

Subgroups

5.16.1.1 Add

SCPI Command :

```
PLATform:EMITter:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PLATform:EMITter:ADD
driver.platform.emitter.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PLATform:EMITter:ADD
driver.platform.emitter.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.16.1.2 BlankRanges

SCPI Commands :

```
PLATform:EMITter:BLANkranges:CLEar
PLATform:EMITter:BLANkranges:COUNt
PLATform:EMITter:BLANkranges:DELete
PLATform:EMITter:BLANkranges:SELect
PLATform:EMITter:BLANkranges:STARt
PLATform:EMITter:BLANkranges:STOP
```

class BlankRangesCls

BlankRanges commands group definition. 7 total commands, 1 Subgroups, 6 group commands

clear() → None

```
# SCPI: PLATform:EMITter:BLANkranges:CLEar
driver.platform.emitter.blankRanges.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PLATform:EMITter:BLANkranges:CLEar
driver.platform.emitter.blankRanges.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PLATform:EMITter:BLANkranges:DELete
driver.platform.emitter.blankRanges.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: PLATform:EMITter:BLANkranges:COUNt
value: float = driver.platform.emitter.blankRanges.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: PLATform:EMITter:BLANkranges:SELect
value: float = driver.platform.emitter.blankRanges.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_start() → float

```
# SCPI: PLATform:EMITter:BLANkranges:STARt
value: float = driver.platform.emitter.blankRanges.get_start()
```

Sets the start angle for the selected blank range. The reference value (i.e. 0DEG) is the configured 'Azimuth' value for the selected emitter. Use together with method RsPulseSeq.Platform.Emitter.BlankRanges.select. To configure several blank ranges with a single command, you can use PLATform:EMITter:BLANkranges . This approach is more efficient than using several blank range start/stop commands.

return
start: float Range: 0 to 360

get_stop() → float

```
# SCPI: PLATform:EMITter:BLANkranges:STOP
value: float = driver.platform.emitter.blankRanges.get_stop()
```

Sets the stop angle for the selected 'Blank Range'. The reference value (i.e. 0DEG) is the configured 'Azimuth' value for the selected emitter. Use together with method RsPulseSeq.Platform.Emitter.BlankRanges.select. To configure several blank ranges with a single command, you can use PLATform:EMITter:BLANkranges . This approach is more efficient than using several blank range start/stop commands.

return
stop: float Range: 0 to 360

set_select(*select: float*) → None

```
# SCPI: PLATform:EMITter:BLANkranges:SELect
driver.platform.emitter.blankRanges.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_start(*start: float*) → None

```
# SCPI: PLATform:EMITter:BLANkranges:STARt
driver.platform.emitter.blankRanges.set_start(start = 1.0)
```

Sets the start angle for the selected blank range. The reference value (i.e. 0DEG) is the configured 'Azimuth' value for the selected emitter. Use together with method RsPulseSeq.Platform.Emitter.BlankRanges.select. To configure several blank ranges with a single command, you can use PLATform:EMITter:BLANkranges . This approach is more efficient than using several blank range start/stop commands.

param start
float Range: 0 to 360

set_stop(*stop: float*) → None

```
# SCPI: PLATform:EMITter:BLANkranges:STOP
driver.platform.emitter.blankRanges.set_stop(stop = 1.0)
```

Sets the stop angle for the selected 'Blank Range'. The reference value (i.e. 0DEG) is the configured 'Azimuth' value for the selected emitter. Use together with method RsPulseSeq.Platform.Emitter.BlankRanges.select. To configure several blank ranges with a single command, you can use PLATform:EMITter:BLANkranges . This approach is more efficient than using several blank range start/stop commands.

param stop
float Range: 0 to 360

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.platform.emitter.blankRanges.clone()
```

Subgroups

5.16.1.2.1 Add

SCPI Command :

```
PLATform:EMITter:BLANkranges:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PLATform:EMITter:BLANkranges:ADD
driver.platform.emitter.blankRanges.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PLATform:EMITter:BLANkranges:ADD
driver.platform.emitter.blankRanges.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.16.1.3 Direction

SCPI Command :

```
PLATform:EMITter:DIRection:AWAY
```

class DirectionCls

Direction commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_away() → bool

```
# SCPI: PLATform:EMITter:DIRection:AWAY
value: bool = driver.platform.emitter.direction.get_away()
```

This command automatically configures the transmission direction of the selected emitter. No effect if emitter is at origin. Affects emitters whose position (relative to the origin) has been defined by one of the following methods:

INTRO_CMD_HELP: Examples of special characters:

- Set X and Y values
- Set Angle and Radius values

The transmission direction is configured so that it is directly away from the origin.

return
away: ON| OFF| 1| 0

set_away(*away: bool*) → None

```
# SCPI: PLATform:EMITter:DIRection:AWAY
driver.platform.emitter.direction.set_away(away = False)
```

This command automatically configures the transmission direction of the selected emitter. No effect if emitter is at origin. Affects emitters whose position (relative to the origin) has been defined by one of the following methods:

INTRO_CMD_HELP: Examples of special characters:

- Set X and Y values
- Set Angle and Radius values

The transmission direction is configured so that it is directly away from the origin.

param away
ON| OFF| 1| 0

5.17 Plot

class PlotCls

Plot commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.plot.clone()
```

Subgroups

5.17.1 Polar

SCPI Commands :

```
PLOT:POLar:CUT
PLOT:POLar:TYPE
```

class PolarCls

Polar commands group definition. 3 total commands, 1 Subgroups, 2 group commands

set_cut(*cut: PolarCut*) → None

```
# SCPI: PLOT:POLar:CUT
driver.plot.polar.set_cut(cut = enums.PolarCut.XY)
```

Sets the diagram cut.

param cut
XY| YZ

set_type_py(*type_py: PolarType*) → None

```
# SCPI: PLOT:POLar:TYPE
driver.plot.polar.set_type_py(type_py = enums.PolarType.CARTesian)
```

Sets the coordinates of the 2D antenna pattern diagram.

param type_py
POLar| CARTesian

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.plot.polar.clone()
```

Subgroups

5.17.1.1 Log

SCPI Command :

```
PLOT:POLar:LOG:MIN
```

class LogCls

Log commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set_min(*min_py: float*) → None

```
# SCPI: PLOT:POLar:LOG:MIN
driver.plot.polar.log.set_min(min_py = 1.0)
```

Sets the minimum value displayed on the y axis.

param min_py
float

5.18 Plugin

SCPI Commands :

```
PLUGin:CATalog
PLUGin:COMMent
PLUGin:CREate
PLUGin:LOAD
PLUGin:NAME
PLUGin:REMove
PLUGin:SELect
```

class PluginCls

Plugin commands group definition. 13 total commands, 1 Subgroups, 7 group commands

get_catalog() → str

```
# SCPI: PLUGin:CATalog
value: str = driver.plugin.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: PLUGin:COMMent
value: str = driver.plugin.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_name() → str

```
# SCPI: PLUGin:NAME
value: str = driver.plugin.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: PLUGin:SELect
value: str = driver.plugin.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

load(*load: str*) → None

```
# SCPI: PLUGin:LOAD
driver.plugin.load(load = 'abc')
```

Loads the selected DLL file, see also 'Plug-in programming API'.

param load
string File path incl. file name and extension

set_comment(*comment: str*) → None

```
# SCPI: PLUGin:COMMent
driver.plugin.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: PLUGin:CREate
driver.plugin.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: PLUGin:NAME
driver.plugin.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: PLUGin:REMove
driver.plugin.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: PLUGin:SELect
driver.plugin.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.plugin.clone()
```

Subgroups

5.18.1 Module

SCPI Commands :

```
PLUGin:MODule:AUTHor
PLUGin:MODule:COMMent
PLUGin:MODule:DATA
PLUGin:MODule:NAME
PLUGin:MODule:TYPE
PLUGin:MODule:VERSion
```

class ModuleCls

Module commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_author() → str

```
# SCPI: PLUGin:MODule:AUTHor
value: str = driver.plugin.module.get_author()
```

Queries information on the loaded file. The query returns information as specified in the description of the corresponding function in 'Plug-in programming API'. The following are the possible values for the type query.

return
author: REPort| IPM | MOP MOP Plugin for IPM Plugin for REPort Plugin for reports created during the waveform generation

get_comment() → str

```
# SCPI: PLUGin:MODule:COMMent
value: str = driver.plugin.module.get_comment()
```

Queries information on the loaded file. The query returns information as specified in the description of the corresponding function in 'Plug-in programming API'. The following are the possible values for the type query.

return
comment: REPort| IPM | MOP MOP Plugin for IPM Plugin for REPort Plugin for reports created during the waveform generation

get_data() → float

```
# SCPI: PLUGin:MODule:DATA
value: float = driver.plugin.module.get_data()
```

Queries whether the plugin requires data from a data source.

return
data: 0 | 1 0 Data source is not required 1 Data source is required

get_name() → str

```
# SCPI: PLUGin:MODule:NAME
value: str = driver.plugin.module.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_type_py() → ModuleType

```
# SCPI: PLUGin:MODule:TYPE
value: enums.ModuleType = driver.plugin.module.get_type_py()
```

Queries information on the loaded file. The query returns information as specified in the description of the corresponding function in 'Plug-in programming API'. The following are the possible values for the type query.

return
type_py: REPort| IPM | MOP MOP Plugin for IPM Plugin for REPort Plugin for reports created during the waveform generation

get_version() → str

```
# SCPI: PLUGin:MODule:VERSion
value: str = driver.plugin.module.get_version()
```

Queries information on the loaded file. The query returns information as specified in the description of the corresponding function in 'Plug-in programming API'. The following are the possible values for the type query.

return
version: REPort| IPM | MOP MOP Plugin for IPM Plugin for REPort Plugin for reports created during the waveform generation

5.19 Preview

SCPI Command :

```
PREView:POSition
```

class PreviewCls

Preview commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_position() → str

```
# SCPI: PREView:POSition
value: str = driver.preview.get_position()
```

If movement is enabled, queries the current positions of the Tx items.

return
position: string Semicolon-separated string with the format: TIME=time_from_simulation_start; ID=Tx item ID;NAME=Tx item alias name;DIST=distancekm;LEVATT=Level at Rx origindBm;AZI=Azimuthdeg;ELEV=Elevationdeg;N=Northkm;E=Eastkm;H=Heightkm;

5.20 Program

SCPI Command :

```
PROGram:MODE
```

class ProgramCls

Program commands group definition. 24 total commands, 15 Subgroups, 1 group commands

get_mode() → ProgramMode

```
# SCPI: PROGram:MODE
value: enums.ProgramMode = driver.program.get_mode()
```

Selects the operation mode on start-up.

return
mode: DEMO| STANdard| EXPert

set_mode(*mode: ProgramMode*) → None

```
# SCPI: PROGram:MODE
driver.program.set_mode(mode = enums.ProgramMode.DEMO)
```

Selects the operation mode on start-up.

param mode
DEMO| STANdard| EXPert

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.program.clone()
```

Subgroups

5.20.1 Adjustments

SCPI Command :

```
PROGram:ADJustments:ENABle
```

class AdjustmentsCls

Adjustments commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:ADJustments:ENABle
value: bool = driver.program.adjustments.get_enable()
```

Enables using the path loss compensation function.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:ADJustments:ENABle
driver.program.adjustments.set_enable(enable = False)
```

Enables using the path loss compensation function.

param enable
ON| OFF| 1| 0

5.20.2 ClassPy

SCPI Command :

```
PROGram:CLASs:ENABle
```

class ClassPyCls

ClassPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:CLASs:ENABle
value: bool = driver.program.classPy.get_enable()
```

Enables whether the workspace classification level appears in the lower window (restart required) .

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:CLASs:ENABle
driver.program.classPy.set_enable(enable = False)
```

Enables whether the workspace classification level appears in the lower window (restart required) .

param enable
ON| OFF| 1| 0

5.20.3 Comment

SCPI Command :

```
PROGram:COMMent:ENABle
```

class CommentCls

Comment commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:COMMent:ENABle
value: bool = driver.program.comment.get_enable()
```

Add timestamp as comment when creating entries.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:COMMent:ENABle
driver.program.comment.set_enable(enable = False)
```

Add timestamp as comment when creating entries.

param enable
ON| OFF| 1| 0

5.20.4 Gpu

SCPI Command :

```
PROGram:GPU:ENABle
```

class GpuCls

Gpu commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:GPU:ENABle
value: bool = driver.program.gpu.get_enable()
```

Enables the GPU (Graphics Processing Unit) to be used for antenna pattern calculations. Using the GPU accelerates the calculation. Requires a restart.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:GPU:ENABle
driver.program.gpu.set_enable(enable = False)
```

Enables the GPU (Graphics Processing Unit) to be used for antenna pattern calculations. Using the GPU accelerates the calculation. Requires a restart.

param enable
ON| OFF| 1| 0

5.20.5 Hide

SCPI Command :

```
PROGram:HIDE
```

class HideCls

Hide commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PROGram:HIDE
driver.program.hide.set()
```

Minimizes/maximizes the R&S Pulse Sequencer Digital workspace.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PROGram:HIDE
driver.program.hide.set_with_opc()
```

Minimizes/maximizes the R&S Pulse Sequencer Digital workspace.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.20.6 Path

SCPI Commands :

```
PROGram:PATH:CALCulated
PROGram:PATH:INSTall
PROGram:PATH:REPort
PROGram:PATH:VOLatile
```

class PathCls

Path commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_calculated() → str

```
# SCPI: PROGram:PATH:CALCulated
value: str = driver.program.path.get_calculated()
```

Sets the directory that holds the calculated waveforms.

return
calculated: string

get_install() → str

```
# SCPI: PROGram:PATH:INSTall
value: str = driver.program.path.get_install()
```

Queries the storage location for repository files.

return
install: string

get_report() → str

```
# SCPI: PROGram:PATH:REPort
value: str = driver.program.path.get_report()
```

Sets the directory that holds generated reports.

return
report: string

get_volatile() → str

```
# SCPI: PROGram:PATH:VOLatile
value: str = driver.program.path.get_volatile()
```

Sets the directory that holds volatile data.

return
volatile: string

set_calculated(*calculated: str*) → None

```
# SCPI: PROGram:PATH:CALCulated
driver.program.path.set_calculated(calculated = 'abc')
```

Sets the directory that holds the calculated waveforms.

param calculated
string

set_report(*report: str*) → None

```
# SCPI: PROGram:PATH:REPort
driver.program.path.set_report(report = 'abc')
```

Sets the directory that holds generated reports.

param report
string

set_volatile(*volatile: str*) → None

```
# SCPI: PROGram:PATH:VOLatile
driver.program.path.set_volatile(volatile = 'abc')
```

Sets the directory that holds volatile data.

param volatile
string

5.20.7 RamBuff

SCPI Command :

```
PROGram:RAMBuff:SIZE
```

class RamBuffCls

RamBuff commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_size() → BufferSize

```
# SCPI: PROGram:RAMBuff:SIZE
value: enums.BufferSize = driver.program.ramBuff.get_size()
```

Sets the ARB RAM buffer size.

return
size: 16M| 64M| 128M| 256M| 512M| 1G

set_size(*size: BufferSize*) → None

```
# SCPI: PROGram:RAMBuff:SIZE
driver.program.ramBuff.set_size(size = enums.BufferSize._128M)
```

Sets the ARB RAM buffer size.

param size
16M| 64M| 128M| 256M| 512M| 1G

5.20.8 Scenario

class ScenarioCls

Scenario commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.program.scenario.clone()
```

Subgroups

5.20.8.1 Xtrg

SCPI Command :

```
PROGram:SCENario:XTRG:ENABle
```

class XtrgCls

Xtrg commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:SCENario:XTRG:ENABle
value: bool = driver.program.scenario.xtrg.get_enable()
```

Enables using a separate trigger for scenario start, see method RsPulseSeq.Scenario.Trigger.set.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:SCENario:XTRG:ENABle
driver.program.scenario.xtrg.set_enable(enable = False)
```

Enables using a separate trigger for scenario start, see method RsPulseSeq.Scenario.Trigger.set.

param enable
ON| OFF| 1| 0

5.20.9 Settings

class SettingsCls

Settings commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.program.settings.clone()
```

Subgroups

5.20.9.1 Accept

SCPI Command :

```
PROGram:SETTings:ACCept
```

class AcceptCls

Accept commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PROGram:SETTings:ACCept
driver.program.settings.accept.set()
```

Reject changes to program settings.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PROGram:SETTings:ACCept
driver.program.settings.accept.set_with_opc()
```

Reject changes to program settings.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.20.9.2 Reject

SCPI Command :

```
PROGram:SETTings:REJect
```

class RejectCls

Reject commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PROGram:SETTings:REJect
driver.program.settings.reject.set()
```

Reject changes to program settings.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PROGram:SETTings:REJect
driver.program.settings.reject.set_with_opc()
```

Reject changes to program settings.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.20.10 Show

SCPI Command :

```
PROGram:SHOW
```

class ShowCls

Show commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PROGram:SHOW
driver.program.show.set()
```

Minimizes/maximizes the R&S Pulse Sequencer Digital workspace.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PROGram:SHOW
driver.program.show.set_with_opc()
```

Minimizes/maximizes the R&S Pulse Sequencer Digital workspace.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.20.11 Startup

class StartupCls

Startup commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.program.startup.clone()
```

Subgroups

5.20.11.1 Load

SCPI Command :

```
PROGram:STARtup:LOAD:ENABle
```

class LoadCls

Load commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:STARtup:LOAD:ENABle
value: bool = driver.program.startup.load.get_enable()
```

Sets if a scenario is opened each time the software is started up.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:STARtup:LOAD:ENABle
driver.program.startup.load.set_enable(enable = False)
```

Sets if a scenario is opened each time the software is started up.

param enable
ON| OFF| 1| 0

5.20.11.2 Wizard

SCPI Command :

```
PROGram:STARtup:WIZard:ENABle
```

class WizardCls

Wizard commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:STARtup:WIZard:ENABle
value: bool = driver.program.startup.wizard.get_enable()
```

Enable this command, if you wish the wizard to open when the software starts.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:STARtup:WIZard:ENABle
driver.program.startup.wizard.set_enable(enable = False)
```

Enable this command, if you wish the wizard to open when the software starts.

param enable
ON| OFF| 1| 0

5.20.12 StorageLoc

SCPI Command :

```
PROGram:STORageloc:ENABle
```

class StorageLocCls

StorageLoc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:STORageloc:ENABle
value: bool = driver.program.storageLoc.get_enable()
```

If enabled, you can select the directory in that new repository is saved.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:STORageloc:ENABle
driver.program.storageLoc.set_enable(enable = False)
```

If enabled, you can select the directory in that new repository is saved.

param enable
ON| OFF| 1| 0

5.20.13 Toolbar

SCPI Command :

```
PROGram:TOOLbar:ENABle
```

class ToolbarCls

Toolbar commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:TOOLbar:ENABle
value: bool = driver.program.toolbar.get_enable()
```

No command help available

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:TOOLbar:ENABle
driver.program.toolbar.set_enable(enable = False)
```

No command help available

param enable
ON| OFF| 1| 0

5.20.14 Transfer

class TransferCls

Transfer commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.program.transfer.clone()
```

Subgroups

5.20.14.1 Ftp

SCPI Commands :

```
PROGram:TRANsfer:FTP:BLOCksize
PROGram:TRANsfer:FTP:ENABle
PROGram:TRANsfer:FTP:PASSwd
PROGram:TRANsfer:FTP:UNAMe
```

class FtpCls

Ftp commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_block_size() → BlockSize

```
# SCPI: PROGram:TRANsfer:FTP:BLOCksize
value: enums.BlockSize = driver.program.transfer.ftp.get_block_size()
```

Block size used for data transfer.

return
block_size: 16K| 32K| 64K| 1M| 2M| 4M

get_enable() → bool

```
# SCPI: PROGram:TRANsfer:FTP:ENABle
value: bool = driver.program.transfer.ftp.get_enable()
```

Enables an FTP data transfer for transfer of large files to the processing instrument.

return
enable: ON| OFF| 1| 0

get_passwd() → str

```
# SCPI: PROGram:TRANsfer:FTP:PASSwd
value: str = driver.program.transfer.ftp.get_passwd()
```

No command help available

return
passwd: string

get_uname() → str

```
# SCPI: PROGram:TRANsfer:FTP:UNAMe
value: str = driver.program.transfer.ftp.get_uname()
```

Sets the user name and password of the processing instrument.

return
uname: string

set_block_size(*block_size: BlockSize*) → None

```
# SCPI: PROGram:TRANsfer:FTP:BLOCksize
driver.program.transfer.ftp.set_block_size(block_size = enums.BlockSize._16K)
```

Block size used for data transfer.

param block_size
16K| 32K| 64K| 1M| 2M| 4M

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:TRANsfer:FTP:ENABle
driver.program.transfer.ftp.set_enable(enable = False)
```

Enables an FTP data transfer for transfer of large files to the processing instrument.

param enable
ON| OFF| 1| 0

set_passwd(*passwd: str*) → None

```
# SCPI: PROGram:TRANsfer:FTP:PASSwd
driver.program.transfer.ftp.set_passwd(passwd = 'abc')
```

No command help available

param passwd
string

set_uname(*uname: str*) → None

```
# SCPI: PROGram:TRANsfer:FTP:UNAMe
driver.program.transfer.ftp.set_uname(uname = 'abc')
```

Sets the user name and password of the processing instrument.

param uname
string

5.20.15 Tutorials

class TutorialsCls

Tutorials commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.program.tutorials.clone()
```

Subgroups

5.20.15.1 Show

SCPI Command :

```
PROGram:TUTorials:SHOW:ENABle
```

class ShowCls

Show commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: PROGram:TUTorials:SHOW:ENABle
value: bool = driver.program.tutorials.show.get_enable()
```

This setting re-enables all tutorials. Tutorials are shown upon opening certain dialogs for the first time (e.g. 2D Map) . When the tutorial has been viewed, it is then disabled.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: PROGram:TUTorials:SHOW:ENABle
driver.program.tutorials.show.set_enable(enable = False)
```

This setting re-enables all tutorials. Tutorials are shown upon opening certain dialogs for the first time (e.g. 2D Map) . When the tutorial has been viewed, it is then disabled.

param enable
ON| OFF| 1| 0

5.21 Pulse

SCPI Commands :

```
PULSe:CATalog
PULSe:COMMent
PULSe:CREate
PULSe:CUSTom
PULSe:NAME
PULSe:REMove
PULSe:SELect
PULSe:SETTings
```

class PulseCls

Pulse commands group definition. 150 total commands, 9 Subgroups, 8 group commands

get_catalog() → str

```
# SCPI: PULSe:CATalog
value: str = driver.pulse.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: PULSe:COMMent
value: str = driver.pulse.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_custom() → bool

```
# SCPI: PULSe:CUSTom
value: bool = driver.pulse.get_custom()
```

Enables the use of a custom envelope function

return
custom: ON| OFF| 1| 0

get_name() → str

```
# SCPI: PULSe:NAME
value: str = driver.pulse.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: PULSe:SELect
value: str = driver.pulse.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_comment(*comment: str*) → None

```
# SCPI: PULSe:COMMent
driver.pulse.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: PULSe:CREate
driver.pulse.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_custom(*custom: bool*) → None

```
# SCPI: PULSe:CUSTom
driver.pulse.set_custom(custom = False)
```

Enables the use of a custom envelope function

param custom
ON| OFF| 1| 0

set_name(*name: str*) → None

```
# SCPI: PULSe:NAME
driver.pulse.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: PULSe:REMove
driver.pulse.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

`set_select`(*select: str*) → None

```
# SCPI: PULSe:SELect
driver.pulse.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

`set_settings`(*settings: PulseSetting*) → None

```
# SCPI: PULSe:SETTings
driver.pulse.set_settings(settings = enums.PulseSetting.GENeral)
```

Switches between the displayed settings.

param settings
TIMing| MOP| MKR| GENeral | | LEVel

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.clone()
```

Subgroups

5.21.1 Envelope

SCPI Commands :

```
PULSe:ENVelope:EQUation
PULSe:ENVelope:MODE
```

`class EnvelopeCls`

Envelope commands group definition. 13 total commands, 1 Subgroups, 2 group commands

`get_equation`() → str

```
# SCPI: PULSe:ENVelope:EQUation
value: str = driver.pulse.envelope.get_equation()
```

Determines the envelope mathematically.

return
equation: string

get_mode() → EnvelopeMode

```
# SCPI: PULSe:ENVelope:MODE
value: enums.EnvelopeMode = driver.pulse.envelope.get_mode()
```

Selects the type of the custom envelope function.

return
mode: DATA| EQUation

set_equation(*equation: str*) → None

```
# SCPI: PULSe:ENVelope:EQUation
driver.pulse.envelope.set_equation(equation = 'abc')
```

Determines the envelope mathematically.

param equation
string

set_mode(*mode: EnvelopeMode*) → None

```
# SCPI: PULSe:ENVelope:MODE
driver.pulse.envelope.set_mode(mode = enums.EnvelopeMode.DATA)
```

Selects the type of the custom envelope function.

param mode
DATA| EQUation

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.envelope.clone()
```

Subgroups

5.21.1.1 Data

SCPI Commands :

```
PULSe:ENVelope:DATA:CLEar
PULSe:ENVelope:DATA:LOAD
PULSe:ENVelope:DATA:MULTiplier
PULSe:ENVelope:DATA:OFFSet
PULSe:ENVelope:DATA:SAVE
PULSe:ENVelope:DATA:UNIT
```

class DataCls

Data commands group definition. 11 total commands, 1 Subgroups, 6 group commands

clear() → None

```
# SCPI: PULSe:ENVelope:DATA:CLEar
driver.pulse.envelope.data.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:ENVelope:DATA:CLEar
driver.pulse.envelope.data.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_load() → str

```
# SCPI: PULSe:ENVelope:DATA:LOAD
value: str = driver.pulse.envelope.data.get_load()
```

Loads an envelope description form an ASCII file.

return
load: string File path, file name, and file extension

get_multiplier() → float

```
# SCPI: PULSe:ENVelope:DATA:MULTiplier
value: float = driver.pulse.envelope.data.get_multiplier()
```

Sets a multiplier factor.

return
multiplier: float Range: -100 to 100

get_offset() → float

```
# SCPI: PULSe:ENVelope:DATA:OFFSet
value: float = driver.pulse.envelope.data.get_offset()
```

Sets an offset for the envelope.

return
offset: float Range: -100 to 100

get_save() → str

```
# SCPI: PULSe:ENVelope:DATA:SAVE
value: str = driver.pulse.envelope.data.get_save()
```

Stores the custom envelope into file.

return
save: string File path, file name, and file extension

get_unit() → DataUnit

```
# SCPI: PULSe:ENVelope:DATA:UNIT
value: enums.DataUnit = driver.pulse.envelope.data.get_unit()
```

Sets the data format.

return
unit: VOLTage| WATTs| DB

set_load(*load: str*) → None

```
# SCPI: PULSe:ENVelope:DATA:LOAD
driver.pulse.envelope.data.set_load(load = 'abc')
```

Loads an envelope description form an ASCII file.

param load
string File path, file name, and file extension

set_multiplier(*multiplier: float*) → None

```
# SCPI: PULSe:ENVelope:DATA:MULTiplier
driver.pulse.envelope.data.set_multiplier(multiplier = 1.0)
```

Sets a multiplier factor.

param multiplier
float Range: -100 to 100

set_offset(*offset: float*) → None

```
# SCPI: PULSe:ENVelope:DATA:OFFSet
driver.pulse.envelope.data.set_offset(offset = 1.0)
```

Sets an offset for the envelope.

param offset
float Range: -100 to 100

set_save(*save: str*) → None

```
# SCPI: PULSe:ENVelope:DATA:SAVE
driver.pulse.envelope.data.set_save(save = 'abc')
```

Stores the custom envelope into file.

param save
string File path, file name, and file extension

set_unit(*unit: DataUnit*) → None

```
# SCPI: PULSe:ENVelope:DATA:UNIT
driver.pulse.envelope.data.set_unit(unit = enums.DataUnit.DB)
```

Sets the data format.

param unit
VOLTage| WATTs| DB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.envelope.data.clone()
```

Subgroups

5.21.1.1.1 Item

SCPI Commands :

```
PULSe:ENVelope:DATA:ITEM:COUNt
PULSe:ENVelope:DATA:ITEM:DELete
PULSe:ENVelope:DATA:ITEM:SELect
PULSe:ENVelope:DATA:ITEM:VALue
```

class ItemCls

Item commands group definition. 5 total commands, 1 Subgroups, 4 group commands

delete(*delete: float*) → None

```
# SCPI: PULSe:ENVelope:DATA:ITEM:DELete
driver.pulse.envelope.data.item.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: PULSe:ENVelope:DATA:ITEM:COUNt
value: float = driver.pulse.envelope.data.item.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: PULSe:ENVelope:DATA:ITEM:SELect
value: float = driver.pulse.envelope.data.item.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → float

```
# SCPI: PULSe:ENVelope:DATA:ITEM:VALue
value: float = driver.pulse.envelope.data.item.get_value()
```

Sets the value of the selected item.

return
value: float Range: -1e+09 to 1e+09

set_select(*select: float*) → None

```
# SCPI: PULSe:ENVelope:DATA:ITEM:SELect
driver.pulse.envelope.data.item.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*value: float*) → None

```
# SCPI: PULSe:ENVelope:DATA:ITEM:VALue
driver.pulse.envelope.data.item.set_value(value = 1.0)
```

Sets the value of the selected item.

param value
float Range: -1e+09 to 1e+09

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.envelope.data.item.clone()
```

Subgroups

5.21.1.1.1.1 Add

SCPI Command :

```
PULSe:ENVelope:DATA:ITEM:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PULSe:ENVelope:DATA:ITEM:ADD
driver.pulse.envelope.data.item.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:ENVelope:DATA:ITEM:ADD
driver.pulse.envelope.data.item.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.21.2 Level

SCPI Commands :

```
PULSe:LEVel:DROop
PULSe:LEVel:OFF
PULSe:LEVel:ON
```

class LevelCls

Level commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_droop() → float

```
# SCPI: PULSe:LEVel:DROop
value: float = driver.pulse.level.get_droop()
```

Sets the amplitude droop.

return
droop: float Range: 0 to 50

get_off() → float

```
# SCPI: PULSe:LEVel:OFF
value: float = driver.pulse.level.get_off()
```

Sets the power during the pulse on time or the pulse off time.

return
off: No help available

get_on() → float

```
# SCPI: PULSe:LEVel:ON
value: float = driver.pulse.level.get_on()
```

Sets the power during the pulse on time or the pulse off time.

return
on: float Range: 0 to 100

set_droop(*droop: float*) → None

```
# SCPI: PULSe:LEVel:DROop
driver.pulse.level.set_droop(droop = 1.0)
```

Sets the amplitude droop.

param droop
float Range: 0 to 50

set_off(*off: float*) → None

```
# SCPI: PULSe:LEVel:OFF
driver.pulse.level.set_off(off = 1.0)
```

Sets the power during the pulse on time or the pulse off time.

param off
float Range: 0 to 100

set_on(*on: float*) → None

```
# SCPI: PULSe:LEVel:ON
driver.pulse.level.set_on(on = 1.0)
```

Sets the power during the pulse on time or the pulse off time.

param on
float Range: 0 to 100

5.21.3 Marker

SCPI Commands :

```
PULSe:MARKer:AUTO
PULSe:MARKer:FALL
PULSe:MARKer:GATE
PULSe:MARKer:POST
PULSe:MARKer:PRE
PULSe:MARKer:RISE
PULSe:MARKer:WIDTh
```

class MarkerCls

Marker commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_auto() → float

```
# SCPI: PULSe:MARKer:AUTO
value: float = driver.pulse.marker.get_auto()
```

Enables up to four restart markers.

return
auto: float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

get_fall() → float

```
# SCPI: PULSe:MARKer:FALL
value: float = driver.pulse.marker.get_fall()
```

Enables up to four restart markers.

return
fall: No help available

get_gate() → float

```
# SCPI: PULSe:MARKer:GATE
value: float = driver.pulse.marker.get_gate()
```

Enables up to four restart markers.

return
gate: No help available

get_post() → float

```
# SCPI: PULSe:MARKer:POST
value: float = driver.pulse.marker.get_post()
```

Enables up to four restart markers.

return
post: No help available

get_pre() → float

```
# SCPI: PULSe:MARKer:PRE
value: float = driver.pulse.marker.get_pre()
```

Enables up to four restart markers.

return
pre: No help available

get_rise() → float

```
# SCPI: PULSe:MARKer:RISE
value: float = driver.pulse.marker.get_rise()
```

Enables up to four restart markers.

return
rise: No help available

get_width() → float

```
# SCPI: PULSe:MARKer:WIDTh
value: float = driver.pulse.marker.get_width()
```

Enables up to four restart markers.

return
width: No help available

set_auto(*auto: float*) → None

```
# SCPI: PULSe:MARKer:AUTO
driver.pulse.marker.set_auto(auto = 1.0)
```

Enables up to four restart markers.

param auto
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_fall(*fall: float*) → None

```
# SCPI: PULSe:MARKer:FALL
driver.pulse.marker.set_fall(fall = 1.0)
```

Enables up to four restart markers.

param fall
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_gate(*gate: float*) → None

```
# SCPI: PULSe:MARKer:GATE
driver.pulse.marker.set_gate(gate = 1.0)
```

Enables up to four restart markers.

param gate
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_post(*post: float*) → None

```
# SCPI: PULSe:MARKer:POST
driver.pulse.marker.set_post(post = 1.0)
```

Enables up to four restart markers.

param post
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_pre(*pre: float*) → None

```
# SCPI: PULSe:MARKer:PRE
driver.pulse.marker.set_pre(pre = 1.0)
```

Enables up to four restart markers.

param pre
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_rise(*rise: float*) → None

```
# SCPI: PULSe:MARKer:RISE
driver.pulse.marker.set_rise(rise = 1.0)
```

Enables up to four restart markers.

param rise
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_width(*width: float*) → None

```
# SCPI: PULSe:MARKer:WIDTh
driver.pulse.marker.set_width(width = 1.0)
```

Enables up to four restart markers.

param width
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

5.21.4 Mop

SCPI Commands :

```
PULSe:MOP:COMMent
PULSe:MOP:ENABle
PULSe:MOP:TYPE
```

class MopCls

Mop commands group definition. 105 total commands, 24 Subgroups, 3 group commands

get_comment() → str

```
# SCPI: PULSe:MOP:COMMent
value: str = driver.pulse.mop.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_enable() → bool

```
# SCPI: PULSe:MOP:ENABle
value: bool = driver.pulse.mop.get_enable()
```

Defines whether a MOP is applied.

return
enable: ON| OFF| 1| 0

get_type_py() → MopType

```
# SCPI: PULSe:MOP:TYPE
value: enums.MopType = driver.pulse.mop.get_type_py()
```

Select the modulation scheme.

return
type_py: AM| ASK| AMSTep| FM| FSK| FMSTep| CHIRp| PCHirp| BARKer| POLYphase| PLISt| BPSK| QPSK| NOISe| PWISechirp| CCHiprp| PSK8| QAM| MSK | NLCHirp| PLUGin

set_comment(*comment: str*) → None

```
# SCPI: PULSe:MOP:COMMent
driver.pulse.mop.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_enable(*enable: bool*) → None

```
# SCPI: PULSe:MOP:ENABle
driver.pulse.mop.set_enable(enable = False)
```

Defines whether a MOP is applied.

param enable
ON| OFF| 1| 0

set_type_py(*type_py: MopType*) → None

```
# SCPI: PULSe:MOP:TYPE
driver.pulse.mop.set_type_py(type_py = enums.MopType.AM)
```

Select the modulation scheme.

param type_py
AM| ASK| AMSTep| FM| FSK| FMSTep| CHIRp| PCHirp| BARKer| POLYphase| PLISt| BPSK| QPSK| NOISe| PWISechirp| CCHiprp| PSK8| QAM| MSK | NLCHirp| PLUGin

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.clone()
```

Subgroups

5.21.4.1 Am

SCPI Commands :

```
PULSe:MOP:AM:FREQuency
PULSe:MOP:AM:MDEPth
PULSe:MOP:AM:TYPE
```

class AmCls

Am commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_frequency() → float

```
# SCPI: PULSe:MOP:AM:FREQuency
value: float = driver.pulse.mop.am.get_frequency()
```

Sets modulation frequency.

return
frequency: float Range: 0.001 to 1e+09

get_mdepth() → float

```
# SCPI: PULSe:MOP:AM:MDEPth
value: float = driver.pulse.mop.am.get_mdepth()
```

Sets the modulation depth.

return
mdepth: float Range: 0 to 100, Unit: percent

get_type_py() → AmType

```
# SCPI: PULSe:MOP:AM:TYPE
value: enums.AmType = driver.pulse.mop.am.get_type_py()
```

Selects the modulation type.

return
type_py: STD| LSB| USB| SB

set_frequency(*frequency: float*) → None

```
# SCPI: PULSe:MOP:AM:FREQuency
driver.pulse.mop.am.set_frequency(frequency = 1.0)
```

Sets modulation frequency.

param frequency
float Range: 0.001 to 1e+09

set_mdepth(*mdepth: float*) → None

```
# SCPI: PULSe:MOP:AM:MDEPth
driver.pulse.mop.am.set_mdepth(mdepth = 1.0)
```

Sets the modulation depth.

param mdepth
float Range: 0 to 100, Unit: percent

set_type_py(*type_py: AmType*) → None

```
# SCPI: PULSe:MOP:AM:TYPE
driver.pulse.mop.am.set_type_py(type_py = enums.AmType.LSB)
```

Selects the modulation type.

param type_py
STD| LSB| USB| SB

5.21.4.2 AmStep

SCPI Commands :

```
PULSe:MOP:AMSTep:CLEar
PULSe:MOP:AMSTep:COUNt
PULSe:MOP:AMSTep:DELete
PULSe:MOP:AMSTep:DURation
PULSe:MOP:AMSTep:INSert
PULSe:MOP:AMSTep:LEVel
PULSe:MOP:AMSTep:SELect
```

class AmStepCls

AmStep commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: PULSe:MOP:AMSTep:CLEar
driver.pulse.mop.amStep.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:AMSTep:CLEar
driver.pulse.mop.amStep.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PULSe:MOP:AMSTep:DELete
driver.pulse.mop.amStep.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: PULSe:MOP:AMSTep:COUNt
value: float = driver.pulse.mop.amStep.get_count()
```

Queries the number of existing items.

return
count: integer

get_duration() → float

```
# SCPI: PULSe:MOP:AMSTep:DURation
value: float = driver.pulse.mop.amStep.get_duration()
```

Sets the step time.

return
duration: float Range: 0 to 3600, Unit: s

get_level() → float

```
# SCPI: PULSe:MOP:AMSTep:LEVel
value: float = driver.pulse.mop.amStep.get_level()
```

Sets the step level.

return
level: float Range: -100 to 0

get_select() → float

```
# SCPI: PULSe:MOP:AMSTep:SELect
value: float = driver.pulse.mop.amStep.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_duration(*duration: float*) → None

```
# SCPI: PULSe:MOP:AMSTep:DURation
driver.pulse.mop.amStep.set_duration(duration = 1.0)
```

Sets the step time.

param duration
float Range: 0 to 3600, Unit: s

set_insert(*insert: float*) → None

```
# SCPI: PULSe:MOP:AMSTep:INSert
driver.pulse.mop.amStep.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_level(*level: float*) → None

```
# SCPI: PULSe:MOP:AMSTep:LEVel
driver.pulse.mop.amStep.set_level(level = 1.0)
```

Sets the step level.

param level
float Range: -100 to 0

set_select(*select: float*) → None

```
# SCPI: PULSe:MOP:AMSTep:SELect
driver.pulse.mop.amStep.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.amStep.clone()
```

Subgroups

5.21.4.2.1 Add

SCPI Command :

```
PULSe:MOP:AMSTep:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PULSe:MOP:AMSTep:ADD
driver.pulse.mop.amStep.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:AMSTep:ADD
driver.pulse.mop.amStep.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.21.4.3 Ask

SCPI Commands :

```
PULSe:MOP:ASK:INVert
PULSe:MOP:ASK:MDEPth
PULSe:MOP:ASK:SRATe
```

class AskCls

Ask commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_invert() → bool

```
# SCPI: PULSe:MOP:ASK:INVert
value: bool = driver.pulse.mop.ask.get_invert()
```

Inverts the modulation.

return
invert: ON| OFF| 1| 0

get_mdepth() → float

```
# SCPI: PULSe:MOP:ASK:MDEPth
value: float = driver.pulse.mop.ask.get_mdepth()
```

Sets the modulation depth.

return
mdepth: float Range: 0 to 100, Unit: percent

get_symbol_rate() → float

```
# SCPI: PULSe:MOP:ASK:SRATe
value: float = driver.pulse.mop.ask.get_symbol_rate()
```

Sets the symbol rate.

return
srate: float Range: 1 to 1e+09

set_invert(*invert: bool*) → None

```
# SCPI: PULSe:MOP:ASK:INVert
driver.pulse.mop.ask.set_invert(invert = False)
```

Inverts the modulation.

param invert
ON| OFF| 1| 0

set_mdepth(*mdepth: float*) → None

```
# SCPI: PULSe:MOP:ASK:MDEPth
driver.pulse.mop.ask.set_mdepth(mdepth = 1.0)
```

Sets the modulation depth.

param mdepth
float Range: 0 to 100, Unit: percent

set_symbol_rate(*srate: float*) → None

```
# SCPI: PULSe:MOP:ASK:SRATe
driver.pulse.mop.ask.set_symbol_rate(srate = 1.0)
```

Sets the symbol rate.

param srate
float Range: 1 to 1e+09

5.21.4.4 Barker

SCPI Commands :

```
PULSe:MOP:BARKer:BLANk
PULSe:MOP:BARKer:CODE
PULSe:MOP:BARKer:TTIMe
```

class BarkerCls

Barker commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_blank() → bool

```
# SCPI: PULSe:MOP:BARKer:BLANk
value: bool = driver.pulse.mop.barker.get_blank()
```

Blanks out the signal during the transition time.

return
blank: ON| OFF| 1| 0

get_code() → BarkerCode

```
# SCPI: PULSe:MOP:BARKer:CODE
value: enums.BarkerCode = driver.pulse.mop.barker.get_code()
```

Selects the code sequence.

return
code: R2A| R2B| R3| R4A| R4B| R5| R7| R11| R13

get_ttime() → float

```
# SCPI: PULSe:MOP:BARKer:TTIMe
value: float = driver.pulse.mop.barker.get_ttime()
```

Sets the transition time.

return
ttime: float Range: 0 to 50, Unit: percent

set_blank(*blank: bool*) → None

```
# SCPI: PULSe:MOP:BARKer:BLANk
driver.pulse.mop.barker.set_blank(blank = False)
```

Blanks out the signal during the transition time.

param blank
ON| OFF| 1| 0

set_code(*code: BarkerCode*) → None

```
# SCPI: PULSe:MOP:BARKer:CODE
driver.pulse.mop.barker.set_code(code = enums.BarkerCode.R11)
```

Selects the code sequence.

param code
R2A| R2B| R3| R4A| R4B| R5| R7| R11| R13

set_ttime(*ttime: float*) → None

```
# SCPI: PULSe:MOP:BARKer:TTIMe
driver.pulse.mop.barker.set_ttime(ttime = 1.0)
```

Sets the transition time.

param ttime
float Range: 0 to 50, Unit: percent

5.21.4.5 Bpsk

SCPI Commands :

```
PULSe:MOP:BPSK:PHASe
PULSe:MOP:BPSK:TTIMe
PULSe:MOP:BPSK:TTYPe
PULSe:MOP:BPSK:TYPE
```

class BpskCls

Bpsk commands group definition. 6 total commands, 1 Subgroups, 4 group commands

get_phase() → float

```
# SCPI: PULSe:MOP:BPSK:PHASe
value: float = driver.pulse.mop.bpsk.get_phase()
```

Sets the phase.

return
phase: float Range: 0.1 to 180, Unit: degree

get_ttime() → float

```
# SCPI: PULSe:MOP:BPSK:TTIMe
value: float = driver.pulse.mop.bpsk.get_ttime()
```

Sets the transition time.

return
ttime: float Range: 0 to 50, Unit: percent

get_ttype() → BpskTtype

```
# SCPI: PULSe:MOP:BPSK:TTYPe
value: enums.BpskTtype = driver.pulse.mop.bpsk.get_ttype()
```

Selects the transition type.

return
ttype: LINear| COSine

get_type_py() → BpskType

```
# SCPI: PULSe:MOP:BPSK:TYPE
value: enums.BpskType = driver.pulse.mop.bpsk.get_type_py()
```

Sets the modulation type.

return
type_py: STANdard| CONStant

set_phase(*phase: float*) → None

```
# SCPI: PULSe:MOP:BPSK:PHASe
driver.pulse.mop.bpsk.set_phase(phase = 1.0)
```

Sets the phase.

param phase
float Range: 0.1 to 180, Unit: degree

set_ttime(*ttime: float*) → None

```
# SCPI: PULSe:MOP:BPSK:TTIMe
driver.pulse.mop.bpsk.set_ttime(ttime = 1.0)
```

Sets the transition time.

param ttime
float Range: 0 to 50, Unit: percent

set_ttype(*ttype: BpskTtype*) → None

```
# SCPI: PULSe:MOP:BPSK:TTYPe
driver.pulse.mop.bpsk.set_ttype(ttype = enums.BpskTtype.COSine)
```

Selects the transition type.

param ttype
LINear| COSine

set_type_py(*type_py: BpskType*) → None

```
# SCPI: PULSe:MOP:BPSK:TYPE
driver.pulse.mop.bpsk.set_type_py(type_py = enums.BpskType.CONStant)
```

Sets the modulation type.

param type_py
STANdard| CONStant

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.bpsk.clone()
```

Subgroups

5.21.4.5.1 SymbolRate

SCPI Commands :

```
PULSe:MOP:BPSK:SRATe:AUTO
PULSe:MOP:BPSK:SRATe
```

class SymbolRateCls

SymbolRate commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_auto() → bool

```
# SCPI: PULSe:MOP:BPSK:SRATe:AUTO
value: bool = driver.pulse.mop.bpsk.symbolRate.get_auto()
```

Enables automatic adjusting of the bits in the pulse width.

return
auto: ON| OFF| 1| 0

get_value() → float

```
# SCPI: PULSe:MOP:BPSK:SRATe
value: float = driver.pulse.mop.bpsk.symbolRate.get_value()
```

Sets the symbol rate.

return
srate: float Range: 1 to 1e+09

set_auto(*auto: bool*) → None

```
# SCPI: PULSe:MOP:BPSK:SRATe:AUTO
driver.pulse.mop.bpsk.symbolRate.set_auto(auto = False)
```

Enables automatic adjusting of the bits in the pulse width.

param auto
ON| OFF| 1| 0

set_value(*srate: float*) → None

```
# SCPI: PULSe:MOP:BPSK:SRATe
driver.pulse.mop.bpsk.symbolRate.set_value(srate = 1.0)
```

Sets the symbol rate.

param srate
float Range: 1 to 1e+09

5.21.4.6 Cchirp

SCPI Commands :

```
PULSe:MOP:CCHirp:CLEar
PULSe:MOP:CCHirp:COUNt
PULSe:MOP:CCHirp:DELete
PULSe:MOP:CCHirp:FREQuency
PULSe:MOP:CCHirp:INSert
PULSe:MOP:CCHirp:SELect
```

class CchirpCls

Cchirp commands group definition. 7 total commands, 1 Subgroups, 6 group commands

clear() → None

```
# SCPI: PULSe:MOP:CCHirp:CLEar
driver.pulse.mop.cchirp.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:CCHirp:CLEar
driver.pulse.mop.cchirp.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PULSe:MOP:CCHirp:DELete
driver.pulse.mop.cchirp.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: PULSe:MOP:CCHirp:COUNt
value: float = driver.pulse.mop.cchirp.get_count()
```

Queries the number of existing items.

return
count: integer

get_frequency() → float

```
# SCPI: PULSe:MOP:CCHirp:FREQuency
value: float = driver.pulse.mop.cchirp.get_frequency()
```

Set the frequency of the custom chirp.

return
frequency: float Range: -1e+09 to 1e+09

get_select() → float

```
# SCPI: PULSe:MOP:CCHirp:SELect
value: float = driver.pulse.mop.cchirp.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_frequency(*frequency: float*) → None

```
# SCPI: PULSe:MOP:CCHirp:FREQuency
driver.pulse.mop.cchirp.set_frequency(frequency = 1.0)
```

Set the frequency of the custom chirp.

param frequency
float Range: -1e+09 to 1e+09

set_insert(*insert: float*) → None

```
# SCPI: PULSe:MOP:CCHirp:INSert
driver.pulse.mop.cchirp.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_select(*select: float*) → None

```
# SCPI: PULSe:MOP:CCHirp:SELect
driver.pulse.mop.cchirp.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.cchirp.clone()
```

Subgroups

5.21.4.6.1 Add

SCPI Command :

```
PULSe:MOP:CCHirp:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PULSe:MOP:CCHirp:ADD
driver.pulse.mop.cchirp.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:CCHirp:ADD
driver.pulse.mop.cchirp.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.21.4.7 Chirp

SCPI Commands :

```
PULSe:MOP:CHIRp:DEViation
PULSe:MOP:CHIRp:TYPE
```

class ChirpCls

Chirp commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_deviation() → float

```
# SCPI: PULSe:MOP:CHIRp:DEViation
value: float = driver.pulse.mop.chirp.get_deviation()
```

Sets the modulation deviation.

return
deviation: float Range: 1 to 1e+09

get_type_py() → ChirpType

```
# SCPI: PULSe:MOP:CHIRp:TYPE
value: enums.ChirpType = driver.pulse.mop.chirp.get_type_py()
```

Selects the modulation type.

return
type_py: UP| DOWN| SINE| TRIangular| PIECewise

set_deviation(*deviation: float*) → None

```
# SCPI: PULSe:MOP:CHIRp:DEViation
driver.pulse.mop.chirp.set_deviation(deviation = 1.0)
```

Sets the modulation deviation.

param deviation
float Range: 1 to 1e+09

set_type_py(*type_py: ChirpType*) → None

```
# SCPI: PULSe:MOP:CHIRp:TYPE
driver.pulse.mop.chirp.set_type_py(type_py = enums.ChirpType.DOWN)
```

Selects the modulation type.

param type_py
UP| DOWN| SINE| TRIangular| PIECewise

5.21.4.8 Data

SCPI Command :

```
PULSe:MOP:DATA:CODing
```

class DataCls

Data commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_coding() → Coding

```
# SCPI: PULSe:MOP:DATA:CODing
value: enums.Coding = driver.pulse.mop.data.get_coding()
```

Selects the data coding scheme.

return
coding: NONE| DIFFerential| GRAY| DGRay

set_coding(*coding: Coding*) → None

```
# SCPI: PULSe:MOP:DATA:CODing
driver.pulse.mop.data.set_coding(coding = enums.Coding.DGRay)
```

Selects the data coding scheme.

param coding
NONE| DIFFerential| GRAY| DGRay

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.data.clone()
```

Subgroups

5.21.4.8.1 Dsrc

SCPI Commands :

```
PULSe:MOP:DATA:DSRC:RESet
PULSe:MOP:DATA:DSRC
```

class DsrcCls

Dsrc commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_reset() → bool

```
# SCPI: PULSe:MOP:DATA:DSRC:RESet
value: bool = driver.pulse.mop.data.dsrc.get_reset()
```

Resets the data source at the end of the pulse.

return
reset: ON| OFF| 1| 0

get_value() → str

```
# SCPI: PULSe:MOP:DATA:DSRC
value: str = driver.pulse.mop.data.dsrc.get_value()
```

Selects the data source for the modulation, see method RsPulseSeq.Dsrc.create.

return
dsrc: string

set_reset(*reset: bool*) → None

```
# SCPI: PULSe:MOP:DATA:DSRC:RESet
driver.pulse.mop.data.dsrc.set_reset(reset = False)
```

Resets the data source at the end of the pulse.

param reset
ON| OFF| 1| 0

set_value(*dsrc: str*) → None

```
# SCPI: PULSe:MOP:DATA:DSRC
driver.pulse.mop.data.dsrc.set_value(dsrc = 'abc')
```

Selects the data source for the modulation, see method RsPulseSeq.Dsrc.create.

param dsrc
string

5.21.4.9 EightPsk

SCPI Command :

```
PULSe:MOP:8PSK:SRATe
```

class EightPskCls

EightPsk commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_symbol_rate() → float

```
# SCPI: PULSe:MOP:8PSK:SRATe
value: float = driver.pulse.mop.eightPsk.get_symbol_rate()
```

Sets the symbol rate of the modulated signal.

return
srate: float Range: 1 to 1e+09

set_symbol_rate(*srate: float*) → None

```
# SCPI: PULSe:MOP:8PSK:SRATe
driver.pulse.mop.eightPsk.set_symbol_rate(srate = 1.0)
```

Sets the symbol rate of the modulated signal.

param srate
float Range: 1 to 1e+09

5.21.4.10 Exclude

SCPI Commands :

```
PULSe:MOP:EXCLude:ENABle
PULSe:MOP:EXCLude:MODE
```

class ExcludeCls

Exclude commands group definition. 6 total commands, 2 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: PULSe:MOP:EXCLude:ENABle
value: bool = driver.pulse.mop.exclude.get_enable()
```

Activates the restriction of the modulation.

return
enable: ON| OFF| 1| 0

get_mode() → ExcMode

```
# SCPI: PULSe:MOP:EXCLude:MODE
value: enums.ExcMode = driver.pulse.mop.exclude.get_mode()
```

Selects the parameter that determines the area on that the MOP is applied.

return
mode: TIME| LEVel| WIDTh

set_enable(*enable: bool*) → None

```
# SCPI: PULSe:MOP:EXCLude:ENABle
driver.pulse.mop.exclude.set_enable(enable = False)
```

Activates the restriction of the modulation.

param enable
ON| OFF| 1| 0

set_mode(*mode: ExcMode*) → None

```
# SCPI: PULSe:MOP:EXCLude:MODE
driver.pulse.mop.exclude.set_mode(mode = enums.ExcMode.LEVel)
```

Selects the parameter that determines the area on that the MOP is applied.

param mode
TIME| LEVel| WIDTh

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.exclude.clone()
```

Subgroups

5.21.4.10.1 Level

SCPI Commands :

```
PULSe:MOP:EXCLude:LEVel:STARt
PULSe:MOP:EXCLude:LEVel:STOP
```

class LevelCls

Level commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_start() → float

```
# SCPI: PULSe:MOP:EXCLude:LEVel:STARt
value: float = driver.pulse.mop.exclude.level.get_start()
```

Sets the threshold levels at the beginning and the end of a pulse for the modulation to be excluded.

return
start: No help available

get_stop() → float

```
# SCPI: PULSe:MOP:EXCLude:LEVel:STOP
value: float = driver.pulse.mop.exclude.level.get_stop()
```

Sets the threshold levels at the beginning and the end of a pulse for the modulation to be excluded.

return
stop: float Range: 0 to 100

set_start(*start: float*) → None

```
# SCPI: PULSe:MOP:EXCLude:LEVel:STARt
driver.pulse.mop.exclude.level.set_start(start = 1.0)
```

Sets the threshold levels at the beginning and the end of a pulse for the modulation to be excluded.

param start
float Range: 0 to 100

set_stop(*stop: float*) → None

```
# SCPI: PULSe:MOP:EXCLude:LEVel:STOP
driver.pulse.mop.exclude.level.set_stop(stop = 1.0)
```

Sets the threshold levels at the beginning and the end of a pulse for the modulation to be excluded.

param stop
float Range: 0 to 100

5.21.4.10.2 Time

SCPI Commands :

```
PULSe:MOP:EXCLude:TIME:STARt
PULSe:MOP:EXCLude:TIME:STOP
```

class TimeCls

Time commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_start() → float

```
# SCPI: PULSe:MOP:EXCLude:TIME:STARt
value: float = driver.pulse.mop.exclude.time.get_start()
```

Sets a time span to be excluded at the beginning and at the end of the pulse.

return
start: No help available

get_stop() → float

```
# SCPI: PULSe:MOP:EXCLude:TIME:STOP
value: float = driver.pulse.mop.exclude.time.get_stop()
```

Sets a time span to be excluded at the beginning and at the end of the pulse.

return
stop: float Range: 0 to 5e-07

set_start(*start: float*) → None

```
# SCPI: PULSe:MOP:EXCLude:TIME:STARt
driver.pulse.mop.exclude.time.set_start(start = 1.0)
```

Sets a time span to be excluded at the beginning and at the end of the pulse.

param start
float Range: 0 to 5e-07

set_stop(*stop: float*) → None

```
# SCPI: PULSe:MOP:EXCLude:TIME:STOP
driver.pulse.mop.exclude.time.set_stop(stop = 1.0)
```

Sets a time span to be excluded at the beginning and at the end of the pulse.

param stop
float Range: 0 to 5e-07

5.21.4.11 FilterPy

SCPI Commands :

```
PULSe:MOP:FILTer:BT
PULSe:MOP:FILTer:BWIDth
PULSe:MOP:FILTer:LENGth
PULSe:MOP:FILTer:ROLLoff
PULSe:MOP:FILTer:TYPE
```

class FilterPyCls

FilterPy commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_bandwidth() → float

```
# SCPI: PULSe:MOP:FILTer:BWIDth
value: float = driver.pulse.mop.filterPy.get_bandwidth()
```

Sets the transition bandwidth of the filter.

return
bwidth: float Range: 1 to 1e+09

get_bt() → float

```
# SCPI: PULSe:MOP:FILTer:BT
value: float = driver.pulse.mop.filterPy.get_bt()
```

Sets the B x T filter parameter.

return
bt: float Range: 0.15 to 2.5

get_length() → float

```
# SCPI: PULSe:MOP:FILTer:LENGth
value: float = driver.pulse.mop.filterPy.get_length()
```

Sets the filter length.

return
length: integer Range: 1 to 64

get_rolloff() → float

```
# SCPI: PULSe:MOP:FILTer:ROLLoff
value: float = driver.pulse.mop.filterPy.get_rolloff()
```

Sets the roll off factor.

return
rolloff: float Range: 0.05 to 1

get_type_py() → FilterType

```
# SCPI: PULSe:MOP:FILTer:TYPE
value: enums.FilterType = driver.pulse.mop.filterPy.get_type_py()
```

Selects the filter type.

return
type_py: NONE| RECTangular| COS| RCOS| GAUSs| LPASs| FSKGauss| SOQPsk| SMWRect

set_bandwidth(*bwidth: float*) → None

```
# SCPI: PULSe:MOP:FILTer:BWIDth
driver.pulse.mop.filterPy.set_bandwidth(bwidth = 1.0)
```

Sets the transition bandwidth of the filter.

param bwidth
float Range: 1 to 1e+09

set_bt(*bt: float*) → None

```
# SCPI: PULSe:MOP:FILTer:BT
driver.pulse.mop.filterPy.set_bt(bt = 1.0)
```

Sets the B x T filter parameter.

param bt
float Range: 0.15 to 2.5

set_length(*length: float*) → None

```
# SCPI: PULSe:MOP:FILTer:LENGth
driver.pulse.mop.filterPy.set_length(length = 1.0)
```

Sets the filter length.

param length
integer Range: 1 to 64

set_rolloff(*rolloff: float*) → None

```
# SCPI: PULSe:MOP:FILTer:ROLLoff
driver.pulse.mop.filterPy.set_rolloff(rolloff = 1.0)
```

Sets the roll off factor.

param rolloff
float Range: 0.05 to 1

set_type_py(*type_py: FilterType*) → None

```
# SCPI: PULSe:MOP:FILTer:TYPE
driver.pulse.mop.filterPy.set_type_py(type_py = enums.FilterType.COS)
```

Selects the filter type.

param type_py
NONE| RECTangular| COS| RCOS| GAUSs| LPASs| FSKGauss| SOQPsk| SMWRect

5.21.4.12 Fm

SCPI Commands :

```
PULSe:MOP:FM:DEViation
PULSe:MOP:FM:FREQuency
```

class FmCls

Fm commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_deviation() → float

```
# SCPI: PULSe:MOP:FM:DEViation
value: float = driver.pulse.mop.fm.get_deviation()
```

Sets the modulation deviation.

return
deviation: float Range: 0.1 to 1e+09, Unit: Hz

get_frequency() → float

```
# SCPI: PULSe:MOP:FM:FREQuency
value: float = driver.pulse.mop.fm.get_frequency()
```

Sets the modulation frequency.

return
frequency: float Range: 0.002 to 1e+09

set_deviation(*deviation: float*) → None

```
# SCPI: PULSe:MOP:FM:DEViation
driver.pulse.mop.fm.set_deviation(deviation = 1.0)
```

Sets the modulation deviation.

param deviation
float Range: 0.1 to 1e+09, Unit: Hz

set_frequency(*frequency: float*) → None

```
# SCPI: PULSe:MOP:FM:FREQuency
driver.pulse.mop.fm.set_frequency(frequency = 1.0)
```

Sets the modulation frequency.

param frequency
float Range: 0.002 to 1e+09

5.21.4.13 FmStep

SCPI Commands :

```
PULSe:MOP:FMSTep:CLEar
PULSe:MOP:FMSTep:COUNt
PULSe:MOP:FMSTep:DELete
PULSe:MOP:FMSTep:DURation
PULSe:MOP:FMSTep:FREQuency
PULSe:MOP:FMSTep:INSert
PULSe:MOP:FMSTep:SELect
```

class FmStepCls

FmStep commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: PULSe:MOP:FMSTep:CLEar
driver.pulse.mop.fmStep.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:FMSTep:CLEar
driver.pulse.mop.fmStep.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PULSe:MOP:FMSTep:DELete
driver.pulse.mop.fmStep.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: PULSe:MOP:FMSTep:COUNt
value: float = driver.pulse.mop.fmStep.get_count()
```

Queries the number of existing items.

return
count: integer

get_duration() → float

```
# SCPI: PULSe:MOP:FMSTep:DURation
value: float = driver.pulse.mop.fmStep.get_duration()
```

Sets the step time.

return
duration: float Range: 0 to 3600, Unit: s

get_frequency() → float

```
# SCPI: PULSe:MOP:FMSTep:FREQuency
value: float = driver.pulse.mop.fmStep.get_frequency()
```

Sets the step frequency.

return
frequency: float Range: -1e+09 to 1e+09

get_select() → float

```
# SCPI: PULSe:MOP:FMSTep:SELect
value: float = driver.pulse.mop.fmStep.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_duration(*duration: float*) → None

```
# SCPI: PULSe:MOP:FMSTep:DURation
driver.pulse.mop.fmStep.set_duration(duration = 1.0)
```

Sets the step time.

param duration
float Range: 0 to 3600, Unit: s

set_frequency(*frequency: float*) → None

```
# SCPI: PULSe:MOP:FMSTep:FREQuency
driver.pulse.mop.fmStep.set_frequency(frequency = 1.0)
```

Sets the step frequency.

param frequency
float Range: -1e+09 to 1e+09

set_insert(*insert: float*) → None

```
# SCPI: PULSe:MOP:FMSTep:INSert
driver.pulse.mop.fmStep.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_select(*select: float*) → None

```
# SCPI: PULSe:MOP:FMSTep:SELect
driver.pulse.mop.fmStep.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.fmStep.clone()
```

Subgroups

5.21.4.13.1 Add

SCPI Command :

```
PULSe:MOP:FMSTep:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PULSe:MOP:FMSTep:ADD
driver.pulse.mop.fmStep.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:FMSTep:ADD
driver.pulse.mop.fmStep.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.21.4.14 Fsk

SCPI Commands :

```
PULSe:MOP:FSK:DEViation
PULSe:MOP:FSK:INVert
PULSe:MOP:FSK:SRATe
PULSe:MOP:FSK:TYPE
```

class FskCls

Fsk commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_deviation() → float

```
# SCPI: PULSe:MOP:FSK:DEViation
value: float = driver.pulse.mop.fsk.get_deviation()
```

Sets the modulation deviation.

return
deviation: float Range: 0.001 to 1e+09, Unit: Hz

get_invert() → bool

```
# SCPI: PULSe:MOP:FSK:INVert
value: bool = driver.pulse.mop.fsk.get_invert()
```

Inverts the modulation.

return
invert: ON| OFF| 1| 0

get_symbol_rate() → float

```
# SCPI: PULSe:MOP:FSK:SRATe
value: float = driver.pulse.mop.fsk.get_symbol_rate()
```

Sets the symbol rate of the modulated signal.

return
srate: float Range: 1 to 1e+09

get_type_py() → FskType

```
# SCPI: PULSe:MOP:FSK:TYPE
value: enums.FskType = driver.pulse.mop.fsk.get_type_py()
```

Selects the FSK modulation type.

return
type_py: FS2| FS4| FS8| FS16| FS32| FS64

set_deviation(*deviation: float*) → None

```
# SCPI: PULSe:MOP:FSK:DEViation
driver.pulse.mop.fsk.set_deviation(deviation = 1.0)
```

Sets the modulation deviation.

param deviation
float Range: 0.001 to 1e+09, Unit: Hz

set_invert(*invert: bool*) → None

```
# SCPI: PULSe:MOP:FSK:INVert
driver.pulse.mop.fsk.set_invert(invert = False)
```

Inverts the modulation.

param invert
ON| OFF| 1| 0

set_symbol_rate(*srate: float*) → None

```
# SCPI: PULSe:MOP:FSK:SRATe
driver.pulse.mop.fsk.set_symbol_rate(srate = 1.0)
```

Sets the symbol rate of the modulated signal.

param srate
float Range: 1 to 1e+09

set_type_py(*type_py: FskType*) → None

```
# SCPI: PULSe:MOP:FSK:TYPE
driver.pulse.mop.fsk.set_type_py(type_py = enums.FskType.FS16)
```

Selects the FSK modulation type.

param type_py
FS2| FS4| FS8| FS16| FS32| FS64

5.21.4.15 Msk

SCPI Commands :

```
PULSe:MOP:MSK:INVert
PULSe:MOP:MSK:SRATe
```

class MskCls

Msk commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_invert() → bool

```
# SCPI: PULSe:MOP:MSK:INVert
value: bool = driver.pulse.mop.msk.get_invert()
```

Inverts the modulation.

return
invert: ON| OFF| 1| 0

get_symbol_rate() → float

```
# SCPI: PULSe:MOP:MSK:SRATe
value: float = driver.pulse.mop.msk.get_symbol_rate()
```

Sets the symbol rate.

return
srate: float Range: 1 to 1e+09

set_invert(*invert: bool*) → None

```
# SCPI: PULSe:MOP:MSK:INVert
driver.pulse.mop.msk.set_invert(invert = False)
```

Inverts the modulation.

param invert
ON| OFF| 1| 0

set_symbol_rate(*srate: float*) → None

```
# SCPI: PULSe:MOP:MSK:SRATe
driver.pulse.mop.msk.set_symbol_rate(srate = 1.0)
```

Sets the symbol rate.

param srate
float Range: 1 to 1e+09

5.21.4.16 NlCirp

SCPI Command :

```
PULSe:MOP:NLCHirp:EQUation
```

class NlCirpCls

NlCirp commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_equation() → str

```
# SCPI: PULSe:MOP:NLCHirp:EQUation
value: str = driver.pulse.mop.nlCirp.get_equation()
```

Determines the chirp mathematically.

return
equation: string

set_equation(*equation: str*) → None

```
# SCPI: PULSe:MOP:NLCHirp:EQUation
driver.pulse.mop.nlCirp.set_equation(equation = 'abc')
```

Determines the chirp mathematically.

param equation
string

5.21.4.17 Noise

SCPI Command :

```
PULSe:MOP:NOISe:BWIDth
```

class NoiseCls

Noise commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_bandwidth() → float

```
# SCPI: PULSe:MOP:NOISe:BWIDth
value: float = driver.pulse.mop.noise.get_bandwidth()
```

Sets the bandwidth.

return
bwidth: float Range: 1 to 1e+09, Unit: Hz

set_bandwidth(*bwidth: float*) → None

```
# SCPI: PULSe:MOP:NOISe:BWIDth
driver.pulse.mop.noise.set_bandwidth(bwidth = 1.0)
```

Sets the bandwidth.

param bwidth
float Range: 1 to 1e+09, Unit: Hz

5.21.4.18 Pchirp

SCPI Commands :

```
PULSe:MOP:PCHirp:CLEar
PULSe:MOP:PCHirp:COEFficient
PULSe:MOP:PCHirp:COUNt
PULSe:MOP:PCHirp:DELete
PULSe:MOP:PCHirp:INSert
PULSe:MOP:PCHirp:SELect
PULSe:MOP:PCHirp:TERM
```

class PchirpCls

Pchirp commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: PULSe:MOP:PCHirp:CLEar
driver.pulse.mop.pchirp.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:PCHirp:CLEar
driver.pulse.mop.pchirp.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PULSe:MOP:PCHirp:DELete
driver.pulse.mop.pchirp.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_coefficient() → float

```
# SCPI: PULSe:MOP:PCHirp:COEFficient
value: float = driver.pulse.mop.pchirp.get_coefficient()
```

Sets the coefficient of the chirp polynomial.

return
coefficient: float Range: -1e+32 to 1e+32

get_count() → float

```
# SCPI: PULSe:MOP:PCHirp:COUNt
value: float = driver.pulse.mop.pchirp.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: PULSe:MOP:PCHirp:SELect
value: float = driver.pulse.mop.pchirp.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_term() → float

```
# SCPI: PULSe:MOP:PCHirp:TERM
value: float = driver.pulse.mop.pchirp.get_term()
```

Sets the term of the chirp polynomial.

return
term: float Range: 0 to 32

set_coefficient(*coefficient: float*) → None

```
# SCPI: PULSe:MOP:PCHirp:COEFficient
driver.pulse.mop.pchirp.set_coefficient(coefficient = 1.0)
```

Sets the coefficient of the chirp polynomial.

param coefficient
float Range: -1e+32 to 1e+32

set_insert(*insert: float*) → None

```
# SCPI: PULSe:MOP:PCHirp:INSert
driver.pulse.mop.pchirp.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_select(*select: float*) → None

```
# SCPI: PULSe:MOP:PCHirp:SELect
driver.pulse.mop.pchirp.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_term(*term: float*) → None

```
# SCPI: PULSe:MOP:PCHirp:TERM
driver.pulse.mop.pchirp.set_term(term = 1.0)
```

Sets the term of the chirp polynomial.

param term
float Range: 0 to 32

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.pchirp.clone()
```

Subgroups

5.21.4.18.1 Add

SCPI Command :

```
PULSe:MOP:PCHirp:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PULSe:MOP:PCHirp:ADD
driver.pulse.mop.pchirp.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:PCHirp:ADD
driver.pulse.mop.pchirp.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.21.4.19 Piecewise

SCPI Commands :

```
PULSe:MOP:PIECewise:CLEar
PULSe:MOP:PIECewise:COUNt
PULSe:MOP:PIECewise:DELete
PULSe:MOP:PIECewise:DURation
PULSe:MOP:PIECewise:INSert
PULSe:MOP:PIECewise:OFFSet
PULSe:MOP:PIECewise:RATE
PULSe:MOP:PIECewise:SELect
```

class PiecewiseCls

Piecewise commands group definition. 9 total commands, 1 Subgroups, 8 group commands

clear() → None

```
# SCPI: PULSe:MOP:PIECewise:CLEar
driver.pulse.mop.piecewise.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:PIECewise:CLEar
driver.pulse.mop.piecewise.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PULSe:MOP:PIECewise:DELete
driver.pulse.mop.piecewise.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: PULSe:MOP:PIECewise:COUNt
value: float = driver.pulse.mop.piecewise.get_count()
```

Queries the number of existing items.

return
count: integer

get_duration() → float

```
# SCPI: PULSe:MOP:PIECewise:DURation
value: float = driver.pulse.mop.piecewise.get_duration()
```

Set the length of the chirp interval as a percentage of the duration the MOP is applied on.

return
duration: float Range: 0 to 100

get_offset() → float

```
# SCPI: PULSe:MOP:PIECewise:OFFSet
value: float = driver.pulse.mop.piecewise.get_offset()
```

Offsets the start frequency of the chirp.

return
offset: float Range: -1e+09 to 1e+09

get_rate() → float

```
# SCPI: PULSe:MOP:PIECewise:RATE
value: float = driver.pulse.mop.piecewise.get_rate()
```

Set the chirp rate.

return
rate: float Range: -1e+15 to 1e+15, Unit: Hz/s

get_select() → float

```
# SCPI: PULSe:MOP:PIECewise:SELect
value: float = driver.pulse.mop.piecewise.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_duration(*duration: float*) → None

```
# SCPI: PULSe:MOP:PIECewise:DURation
driver.pulse.mop.piecewise.set_duration(duration = 1.0)
```

Set the length of the chirp interval as a percentage of the duration the MOP is applied on.

param duration
float Range: 0 to 100

set_insert(*insert: float*) → None

```
# SCPI: PULSe:MOP:PIECewise:INSert
driver.pulse.mop.piecewise.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_offset(*offset: float*) → None

```
# SCPI: PULSe:MOP:PIECewise:OFFSet
driver.pulse.mop.piecewise.set_offset(offset = 1.0)
```

Offsets the start frequency of the chirp.

param offset
float Range: -1e+09 to 1e+09

set_rate(*rate: float*) → None

```
# SCPI: PULSe:MOP:PIECewise:RATE
driver.pulse.mop.piecewise.set_rate(rate = 1.0)
```

Set the chirp rate.

param rate
float Range: -1e+15 to 1e+15, Unit: Hz/s

set_select(*select: float*) → None

```
# SCPI: PULSe:MOP:PIECewise:SELect
driver.pulse.mop.piecewise.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.piecewise.clone()
```

Subgroups

5.21.4.19.1 Add

SCPI Command :

```
PULSe:MOP:PIECewise:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PULSe:MOP:PIECewise:ADD
driver.pulse.mop.piecewise.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:PIECewise:ADD
driver.pulse.mop.piecewise.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.21.4.20 Plist

SCPI Commands :

```
PULSe:MOP:PLISt:CLEar
PULSe:MOP:PLISt:COUNt
PULSe:MOP:PLISt:DELete
PULSe:MOP:PLISt:INSert
PULSe:MOP:PLISt:SELect
PULSe:MOP:PLISt:VALue
```

class PlistCls

Plist commands group definition. 7 total commands, 1 Subgroups, 6 group commands

clear() → None

```
# SCPI: PULSe:MOP:PLISt:CLEar
driver.pulse.mop.plist.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:PLISt:CLEar
driver.pulse.mop.plist.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: PULSe:MOP:PLISt:DELete
driver.pulse.mop.plist.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: PULSe:MOP:PLISt:COUNt
value: float = driver.pulse.mop.plist.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: PULSe:MOP:PLISt:SELect
value: float = driver.pulse.mop.plist.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → float

```
# SCPI: PULSe:MOP:PLISt:VALue
value: float = driver.pulse.mop.plist.get_value()
```

Sets the phase.

return
value: float Range: -180 to 180, Unit: degree

set_insert(*insert: float*) → None

```
# SCPI: PULSe:MOP:PLISt:INSert
driver.pulse.mop.plist.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_select(*select: float*) → None

```
# SCPI: PULSe:MOP:PLISt:SELect
driver.pulse.mop.plist.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*value: float*) → None

```
# SCPI: PULSe:MOP:PLISt:VALue
driver.pulse.mop.plist.set_value(value = 1.0)
```

Sets the phase.

param value
float Range: -180 to 180, Unit: degree

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.plist.clone()
```

Subgroups

5.21.4.20.1 Add

SCPI Command :

```
PULSe:MOP:PLISt:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: PULSe:MOP:PLISt:ADD
driver.pulse.mop.plist.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:PLISt:ADD
driver.pulse.mop.plist.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.21.4.21 Plugin

SCPI Command :

```
PULSe:MOP:PLUGin:NAME
```

class PluginCls

Plugin commands group definition. 6 total commands, 1 Subgroups, 1 group commands

get_name() → str

```
# SCPI: PULSe:MOP:PLUGin:NAME
value: str = driver.pulse.mop.plugin.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: PULSe:MOP:PLUGin:NAME
driver.pulse.mop.plugin.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.plugin.clone()
```

Subgroups

5.21.4.21.1 Variable

SCPI Commands :

```
PULSe:MOP:PLUGin:VARiable:CATalog
PULSe:MOP:PLUGin:VARiable:RESet
PULSe:MOP:PLUGin:VARiable:VALue
```

class VariableCls

Variable commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_catalog() → str

```
# SCPI: PULSe:MOP:PLUGin:VARiable:CATalog
value: str = driver.pulse.mop.plugin.variable.get_catalog()
```

Queries the variables used in the plugin.

return
catalog: string

get_value() → str

```
# SCPI: PULSe:MOP:PLUGin:VARiable:VALue
value: str = driver.pulse.mop.plugin.variable.get_value()
```

Sets the values of the selected variable.

return
value: string

reset() → None

```
# SCPI: PULSe:MOP:PLUGin:VARiable:RESet
driver.pulse.mop.plugin.variable.reset()
```

Resets the variable values to the defaults.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: PULSe:MOP:PLUGin:VARiable:RESet
driver.pulse.mop.plugin.variable.reset_with_opc()
```

Resets the variable values to the defaults.

Same as reset, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*value: str*) → None

```
# SCPI: PULSe:MOP:PLUGin:VARiable:VALue
driver.pulse.mop.plugin.variable.set_value(value = 'abc')
```

Sets the values of the selected variable.

param value
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.plugin.variable.clone()
```

Subgroups

5.21.4.21.1.1 Select

SCPI Commands :

```
PULSe:MOP:PLUGin:VARiable:SELect:ID
PULSe:MOP:PLUGin:VARiable:SELect
```

class SelectCls

Select commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_id() → float

```
# SCPI: PULSe:MOP:PLUGin:VARiable:SELect:ID
value: float = driver.pulse.mop.plugin.variable.select.get_id()
```

Selects a plugin variable ID.

return
idn: float

get_value() → str

```
# SCPI: PULSe:MOP:PLUGin:VARiable:SELect
value: str = driver.pulse.mop.plugin.variable.select.get_value()
```

Selects a plugin variable.

return
select: string

set_id(*idn: float*) → None

```
# SCPI: PULSe:MOP:PLUGin:VARiable:SELect:ID
driver.pulse.mop.plugin.variable.select.set_id(idn = 1.0)
```

Selects a plugin variable ID.

param idn
float

set_value(*select: str*) → None

```
# SCPI: PULSe:MOP:PLUGin:VARiable:SELect
driver.pulse.mop.plugin.variable.select.set_value(select = 'abc')
```

Selects a plugin variable.

param select
string

5.21.4.22 Poly

SCPI Commands :

```
PULSe:MOP:POLY:LENGth
PULSe:MOP:POLY:TYPE
```

class PolyCls

Poly commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_length() → float

```
# SCPI: PULSe:MOP:POLY:LENGth
value: float = driver.pulse.mop.poly.get_length()
```

Sets the polyphase length (code order) .

return
length: integer Range: 1 to 100

get_type_py() → PolynomType

```
# SCPI: PULSe:MOP:POLY:TYPE
value: enums.PolynomType = driver.pulse.mop.poly.get_type_py()
```

Selects the modulation type.

return
type_py: FRANk| P1| P2| P3| P4

set_length(*length: float*) → None

```
# SCPI: PULSe:MOP:POLY:LENGth
driver.pulse.mop.poly.set_length(length = 1.0)
```

Sets the polyphase length (code order) .

param length
integer Range: 1 to 100

set_type_py(*type_py: PolynomType*) → None

```
# SCPI: PULSe:MOP:POLY:TYPE
driver.pulse.mop.poly.set_type_py(type_py = enums.PolynomType.FRANk)
```

Selects the modulation type.

param type_py
FRANk| P1| P2| P3| P4

5.21.4.23 Qam

SCPI Commands :

```
PULSe:MOP:QAM:SRATe
PULSe:MOP:QAM:TYPE
```

class QamCls

Qam commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_symbol_rate() → float

```
# SCPI: PULSe:MOP:QAM:SRATe
value: float = driver.pulse.mop.qam.get_symbol_rate()
```

Sets the symbol rate of the modulated signal.

return
srate: float Range: 1 to 1e+09

get_type_py() → QamType

```
# SCPI: PULSe:MOP:QAM:TYPE
value: enums.QamType = driver.pulse.mop.qam.get_type_py()
```

Selects the QAM type.

return
type_py: Q16| Q32| Q64| Q128| Q256

set_symbol_rate(*srate: float*) → None

```
# SCPI: PULSe:MOP:QAM:SRATe
driver.pulse.mop.qam.set_symbol_rate(srate = 1.0)
```

Sets the symbol rate of the modulated signal.

param srate
float Range: 1 to 1e+09

set_type_py(*type_py: QamType*) → None

```
# SCPI: PULSe:MOP:QAM:TYPE
driver.pulse.mop.qam.set_type_py(type_py = enums.QamType.Q128)
```

Selects the QAM type.

param type_py
Q16| Q32| Q64| Q128| Q256

5.21.4.24 Qpsk

SCPI Commands :

```
PULSe:MOP:QPSK:SRATe
PULSe:MOP:QPSK:TYPE
```

class QpskCls

Qpsk commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_symbol_rate() → float

```
# SCPI: PULSe:MOP:QPSK:SRATe
value: float = driver.pulse.mop.qpsk.get_symbol_rate()
```

Sets the symbol rate.

return
srate: float Range: 1 to 1e+09

get_type_py() → QpskType

```
# SCPI: PULSe:MOP:QPSK:TYPE
value: enums.QpskType = driver.pulse.mop.qpsk.get_type_py()
```

Selects the modulation type.

return
type_py: NORMal| OQPSk| DQPSk| ASOQpsk| BSOQpsk| TGSoqpsk

set_symbol_rate(*srate: float*) → None

```
# SCPI: PULSe:MOP:QPSK:SRATe
driver.pulse.mop.qpsk.set_symbol_rate(srate = 1.0)
```

Sets the symbol rate.

param srate
float Range: 1 to 1e+09

set_type_py(*type_py: QpskType*) → None

```
# SCPI: PULSe:MOP:QPSK:TYPE
driver.pulse.mop.qpsk.set_type_py(type_py = enums.QpskType.ASOQpsk)
```

Selects the modulation type.

param type_py
NORMal| OQPSk| DQPSk| ASOQpsk| BSOQpsk| TGSoqpsk

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pulse.mop.qpsk.clone()
```

Subgroups

5.21.4.24.1 SoQpsk

SCPI Command :

```
PULSe:MOP:QPSK:SOQPsk:IRIG
```

class SoQpskCls

SoQpsk commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_irig() → bool

```
# SCPI: PULSe:MOP:QPSK:SOQPsk:IRIG
value: bool = driver.pulse.mop.qpsk.soQpsk.get_irig()
```

Enables differential encoding according to the telemetry standard IRIG 106-04.

return
irig: ON| OFF| 1| 0

set_irig(*irig: bool*) → None

```
# SCPI: PULSe:MOP:QPSK:SOQPsk:IRIG
driver.pulse.mop.qpsk.soQpsk.set_irig(irig = False)
```

Enables differential encoding according to the telemetry standard IRIG 106-04.

param irig
ON| OFF| 1| 0

5.21.5 Overshoot

SCPI Commands :

```
PULSe:OVERshoot:DECay
PULSe:OVERshoot
```

class OvershootCls

Overshoot commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_decay() → float

```
# SCPI: PULSe:OVERshoot:DECay
value: float = driver.pulse.overshoot.get_decay()
```

Sets the number of peaks.

return
decay: float Range: 1 to 100

get_value() → float

```
# SCPI: PULSe:OVERshoot
value: float = driver.pulse.overshoot.get_value()
```

Sets the overshoot level value.

return
overshoot: float Range: 0 to 50

set_decay(*decay: float*) → None

```
# SCPI: PULSe:OVERshoot:DECay
driver.pulse.overshoot.set_decay(decay = 1.0)
```

Sets the number of peaks.

param decay
float Range: 1 to 100

set_value(*overshoot: float*) → None

```
# SCPI: PULSe:OVERshoot
driver.pulse.overshoot.set_value(overshoot = 1.0)
```

Sets the overshoot level value.

param overshoot
float Range: 0 to 50

5.21.6 Preview

SCPI Commands :

```
PULSe:PREView:MODE
PULSe:PREView:MOP
```

class PreviewCls

Preview commands group definition. 2 total commands, 0 Subgroups, 2 group commands

set_mode(*mode: PreviewMode*) → None

```
# SCPI: PULSe:PREView:MODE
driver.pulse.preview.set_mode(mode = enums.PreviewMode.ENVelope)
```

Switches between the envelope and modulation graphs.

param mode
ENVelope| MOP

set_mop(*mop: PreviewMop*) → None

```
# SCPI: PULSe:PREView:MOP
driver.pulse.preview.set_mop(mop = enums.PreviewMop.FREQuency)
```

Sets the displayed modulation characteristics.

param mop
IQ| PHASe| FREQuency

5.21.7 Ripple

SCPI Commands :

```
PULSe:RIPPle:FREQuency
PULSe:RIPPle
```

class RippleCls

Ripple commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_frequency() → float

```
# SCPI: PULSe:RIPPle:FREQuency
value: float = driver.pulse.ripple.get_frequency()
```

Sets the ripple frequency.

return
frequency: float Range: 0 to 3e+08, Unit: Hz

get_value() → float

```
# SCPI: PULSe:RIPPle
value: float = driver.pulse.ripple.get_value()
```

Sets the ripple level.

return
ripple: float Range: 0 to 50

set_frequency(*frequency: float*) → None

```
# SCPI: PULSe:RIPPle:FREQuency
driver.pulse.ripple.set_frequency(frequency = 1.0)
```

Sets the ripple frequency.

param frequency
float Range: 0 to 3e+08, Unit: Hz

set_value(*ripple: float*) → None

```
# SCPI: PULSe:RIPPle
driver.pulse.ripple.set_value(ripple = 1.0)
```

Sets the ripple level.

param ripple
float Range: 0 to 50

5.21.8 Time

SCPI Commands :

```
PULSe:TIME:FALL
PULSe:TIME:POST
PULSe:TIME:PRE
PULSe:TIME:REFerence
PULSe:TIME:RISE
PULSe:TIME:WIDTh
```

class TimeCls

Time commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_fall() → float

```
# SCPI: PULSe:TIME:FALL
value: float = driver.pulse.time.get_fall()
```

Sets the transition time of the rising and falling edges.

return
fall: No help available

get_post() → float

```
# SCPI: PULSe:TIME:POST
value: float = driver.pulse.time.get_post()
```

Sets the marker's duration.

return
post: float Range: 0 to 3600

get_pre() → float

```
# SCPI: PULSe:TIME:PRE
value: float = driver.pulse.time.get_pre()
```

Sets the marker's duration. Value different than zero shifts the start of the pulse rising edge and the entire pulse.

return
pre: float Range: 0 to 3600

get_reference() → TimeReference

```
# SCPI: PULSe:TIME:REFerence
value: enums.TimeReference = driver.pulse.time.get_reference()
```

Selects a predefined envelope profile.

return
reference: VOLTage| POWer| FULL

get_rise() → float

```
# SCPI: PULSe:TIME:RISE
value: float = driver.pulse.time.get_rise()
```

Sets the transition time of the rising and falling edges.

return
rise: float Range: 0 to 3600

get_width() → float

```
# SCPI: PULSe:TIME:WIDTh
value: float = driver.pulse.time.get_width()
```

Sets the time during that the pulse is on top power.

return
width: float Range: 0 to 3600, Unit: s

set_fall(*fall: float*) → None

```
# SCPI: PULSe:TIME:FALL
driver.pulse.time.set_fall(fall = 1.0)
```

Sets the transition time of the rising and falling edges.

param fall
float Range: 0 to 3600

set_post(*post: float*) → None

```
# SCPI: PULSe:TIME:POST
driver.pulse.time.set_post(post = 1.0)
```

Sets the marker's duration.

param post
float Range: 0 to 3600

set_pre(*pre: float*) → None

```
# SCPI: PULSe:TIME:PRE
driver.pulse.time.set_pre(pre = 1.0)
```

Sets the marker's duration. Value different than zero shifts the start of the pulse rising edge and the entire pulse.

param pre
float Range: 0 to 3600

set_reference(*reference: TimeReference*) → None

```
# SCPI: PULSe:TIME:REFerence
driver.pulse.time.set_reference(reference = enums.TimeReference.FULL)
```

Selects a predefined envelope profile.

param reference
VOLTage| POWer| FULL

set_rise(*rise: float*) → None

```
# SCPI: PULSe:TIME:RISE
driver.pulse.time.set_rise(rise = 1.0)
```

Sets the transition time of the rising and falling edges.

param rise
float Range: 0 to 3600

set_width(*width: float*) → None

```
# SCPI: PULSe:TIME:WIDTh
driver.pulse.time.set_width(width = 1.0)
```

Sets the time during that the pulse is on top power.

param width
float Range: 0 to 3600, Unit: s

5.21.9 TypePy

SCPI Commands :

```
PULSe:TYPE:FALL
PULSe:TYPE:RISE
```

class TypePyCls

TypePy commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_fall() → PulseType

```
# SCPI: PULSe:TYPE:FALL
value: enums.PulseType = driver.pulse.typePy.get_fall()
```

Sets the slope type for the rising and falling edges.

return
fall: No help available

get_rise() → PulseType

```
# SCPI: PULSe:TYPE:RISE
value: enums.PulseType = driver.pulse.typePy.get_rise()
```

Sets the slope type for the rising and falling edges.

return
rise: LINear| COSine| RCOSine| SQRT

set_fall(*fall: PulseType*) → None

```
# SCPI: PULSe:TYPE:FALL
driver.pulse.typePy.set_fall(fall = enums.PulseType.COSine)
```

Sets the slope type for the rising and falling edges.

param fall
LINear| COSine| RCOSine| SQRT

set_rise(*rise: PulseType*) → None

```
# SCPI: PULSe:TYPE:RISE
driver.pulse.typePy.set_rise(rise = enums.PulseType.COSine)
```

Sets the slope type for the rising and falling edges.

param rise
LINear| COSine| RCOSine| SQRT

5.22 Receiver

SCPI Commands :

```
RECeiver:CATalog
RECeiver:COMMent
RECeiver:CREate
RECeiver:MODel
RECeiver:NAME
RECeiver:REMove
RECeiver:SELect
```

class ReceiverCls

Receiver commands group definition. 24 total commands, 1 Subgroups, 7 group commands

get_catalog() → str

```
# SCPI: RECeiver:CATalog
value: str = driver.receiver.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: RECeiver:COMMent
value: str = driver.receiver.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_model() → RecModel

```
# SCPI: RECeiver:MODel
value: enums.RecModel = driver.receiver.get_model()
```

Sets the receiver model.

return
model: INTerfero| TDOA| COMBined For details, see 'Model'. INTerfero Interferometer Calculates the relative phase difference between the single antenna ports. TDOA Time difference of arrival Calculates the absolute time of arrival (TOA) of the incoming signal for each antenna. COMBined Calculates the relative phases between the antenna ports and calculates the the individual TOAs for each antenna port.

get_name() → str

```
# SCPI: RECeiver:NAME
value: str = driver.receiver.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: RECeiver:SELect
value: str = driver.receiver.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_comment(*comment: str*) → None

```
# SCPI: RECeiver:COMMent
driver.receiver.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: RECeiver:CREate
driver.receiver.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_model(*model: RecModel*) → None

```
# SCPI: RECeiver:MODel
driver.receiver.set_model(model = enums.RecModel.COMBined)
```

Sets the receiver model.

param model
INTerfero| TDOA| COMBined For details, see 'Model'. INTerfero Interferometer Calculates the relative phase difference between the single antenna ports. TDOA Time difference of arrival Calculates the absolute time of arrival (TOA) of the incoming signal for each antenna. COMBined Calculates the relative phases between the antenna ports and calculates the the individual TOAs for each antenna port.

set_name(*name: str*) → None

```
# SCPI: RECeiver:NAME
driver.receiver.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: RECeiver:REMove
driver.receiver.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: RECeiver:SELect
driver.receiver.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.receiver.clone()
```

Subgroups

5.22.1 Antenna

SCPI Commands :

```
RECeiver:ANTenna:ALIas
RECeiver:ANTenna:BM
RECeiver:ANTenna:CLEar
RECeiver:ANTenna:DELete
RECeiver:ANTenna:GAIN
RECeiver:ANTenna:PATTern
RECeiver:ANTenna:SCAN
RECeiver:ANTenna:SELect
```

class AntennaCls

Antenna commands group definition. 17 total commands, 3 Subgroups, 8 group commands

clear() → None

```
# SCPI: RECeiver:ANTenna:CLEar
driver.receiver.antenna.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: RECeiver:ANTenna:CLEar
driver.receiver.antenna.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete() → None

```
# SCPI: RECeiver:ANTenna:DELete
driver.receiver.antenna.delete()
```

Deletes the particular item.

delete_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: RECeiver:ANTenna:DELete
driver.receiver.antenna.delete_with_opc()
```

Deletes the particular item.

Same as delete, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_alias() → str

```
# SCPI: RECeiver:ANTenna:ALIas
value: str = driver.receiver.antenna.get_alias()
```

Sets an alias name for the selected antenna element.

return
alias: string

get_bm() → str

```
# SCPI: RECeiver:ANTenna:BM
value: str = driver.receiver.antenna.get_bm()
```

No command help available

return
bm: No help available

get_gain() → float

```
# SCPI: RECeiver:ANTenna:GAIN
value: float = driver.receiver.antenna.get_gain()
```

Sets the gain of the individual antenna element.

return
gain: float Range: -120 to 120

get_pattern() → str

```
# SCPI: RECeiver:ANTenna:PATTern
value: str = driver.receiver.antenna.get_pattern()
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

return
pattern: string

get_scan() → str

```
# SCPI: RECeiver:ANTenna:SCAN
value: str = driver.receiver.antenna.get_scan()
```

Sets the antenna scan.

return
scan: string

get_select() → float

```
# SCPI: RECeiver:ANTenna:SELect
value: float = driver.receiver.antenna.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_alias(*alias: str*) → None

```
# SCPI: RECeiver:ANTenna:ALIas
driver.receiver.antenna.set_alias(alias = 'abc')
```

Sets an alias name for the selected antenna element.

param alias
string

set_bm(*bm: str*) → None

```
# SCPI: RECeiver:ANTenna:BM
driver.receiver.antenna.set_bm(bm = 'abc')
```

No command help available

param bm
No help available

set_gain(*gain: float*) → None

```
# SCPI: RECeiver:ANTenna:GAIN
driver.receiver.antenna.set_gain(gain = 1.0)
```

Sets the gain of the individual antenna element.

param gain
float Range: -120 to 120

set_pattern(*pattern: str*) → None

```
# SCPI: RECeiver:ANTenna:PATTern
driver.receiver.antenna.set_pattern(pattern = 'abc')
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

param pattern
string

set_scan(*scan: str*) → None

```
# SCPI: RECeiver:ANTenna:SCAN
driver.receiver.antenna.set_scan(scan = 'abc')
```

Sets the antenna scan.

param scan
string

set_select(*select: float*) → None

```
# SCPI: RECeiver:ANTenna:SELect
driver.receiver.antenna.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.receiver.antenna.clone()
```

Subgroups

5.22.1.1 Add

SCPI Command :

```
RECeiver:ANTenna:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: RECeiver:ANTenna:ADD
driver.receiver.antenna.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: RECeiver:ANTenna:ADD
driver.receiver.antenna.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.22.1.2 Direction

SCPI Commands :

```
RECeiver:ANTenna:DIRection:AWAY
RECeiver:ANTenna:DIRection:AZIMuth
RECeiver:ANTenna:DIRection:ELEVation
```

class DirectionCls

Direction commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_away() → bool

```
# SCPI: RECeiver:ANTenna:DIRection:AWAY
value: bool = driver.receiver.antenna.direction.get_away()
```

Sets the azimuth automatically, so that the beam axis is radial to the receiver origin.

return
away: ON| OFF| 1| 0

get_azimuth() → float

```
# SCPI: RECeiver:ANTenna:DIRection:AZIMuth
value: float = driver.receiver.antenna.direction.get_azimuth()
```

Turns the antenna beam axis.

return
azimuth: float Range: 0 to 360

get_elevation() → float

```
# SCPI: RECeiver:ANTenna:DIRection:ELEVation
value: float = driver.receiver.antenna.direction.get_elevation()
```

Turns the antenna beam axis.

return
elevation: float Range: -90 to 90

set_away(*away: bool*) → None

```
# SCPI: RECeiver:ANTenna:DIRection:AWAY
driver.receiver.antenna.direction.set_away(away = False)
```

Sets the azimuth automatically, so that the beam axis is radial to the receiver origin.

param away
ON| OFF| 1| 0

set_azimuth(*azimuth: float*) → None

```
# SCPI: RECeiver:ANTenna:DIRection:AZIMuth
driver.receiver.antenna.direction.set_azimuth(azimuth = 1.0)
```

Turns the antenna beam axis.

param azimuth
float Range: 0 to 360

set_elevation(*elevation: float*) → None

```
# SCPI: RECeiver:ANTenna:DIRection:ELEVation
driver.receiver.antenna.direction.set_elevation(elevation = 1.0)
```

Turns the antenna beam axis.

param elevation
float Range: -90 to 90

5.22.1.3 Position

SCPI Commands :

```
RECeiver:ANTenna:POSition:ANGLe
RECeiver:ANTenna:POSition:HEIGht
RECeiver:ANTenna:POSition:RADius
RECeiver:ANTenna:POSition:X
RECeiver:ANTenna:POSition:Y
```

class PositionCls

Position commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_angle() → float

```
# SCPI: RECeiver:ANTenna:POSition:ANGLe
value: float = driver.receiver.antenna.position.get_angle()
```

Sets the antenna element position as an angle offset from the X-axis.

return
angle: float Range: 0 to 360

get_height() → float

```
# SCPI: RECeiver:ANTenna:POSition:HEIGht
value: float = driver.receiver.antenna.position.get_height()
```

Sets the antenna element height, relative to the receiver origin.

return
height: float Range: -1e+06 to 1e+06

get_radius() → float

```
# SCPI: RECeiver:ANTenna:POSition:RADius
value: float = driver.receiver.antenna.position.get_radius()
```

Sets the distance from the antenna element to the receiver origin.

return
radius: float Range: 0 to 1e+06

get_x() → float

```
# SCPI: RECeiver:ANTenna:POSition:X
value: float = driver.receiver.antenna.position.get_x()
```

Sets the antenna element position as X and Y values, relative to the receiver origin.

return
x: No help available

get_y() → float

```
# SCPI: RECeiver:ANTenna:POSition:Y
value: float = driver.receiver.antenna.position.get_y()
```

Sets the antenna element position as X and Y values, relative to the receiver origin.

return
y: float Range: -1e+06 to 1e+06

set_angle(*angle: float*) → None

```
# SCPI: RECeiver:ANTenna:POSition:ANGLe
driver.receiver.antenna.position.set_angle(angle = 1.0)
```

Sets the antenna element position as an angle offset from the X-axis.

param angle
float Range: 0 to 360

set_height(*height: float*) → None

```
# SCPI: RECeiver:ANTenna:POSition:HEIGht
driver.receiver.antenna.position.set_height(height = 1.0)
```

Sets the antenna element height, relative to the receiver origin.

param height
float Range: -1e+06 to 1e+06

set_radius(*radius: float*) → None

```
# SCPI: RECeiver:ANTenna:POSition:RADius
driver.receiver.antenna.position.set_radius(radius = 1.0)
```

Sets the distance from the antenna element to the receiver origin.

param radius
float Range: 0 to 1e+06

set_x(*x: float*) → None

```
# SCPI: RECeiver:ANTenna:POSition:X
driver.receiver.antenna.position.set_x(x = 1.0)
```

Sets the antenna element position as X and Y values, relative to the receiver origin.

param x
float Range: -1e+06 to 1e+06

set_y(*y: float*) → None

```
# SCPI: RECeiver:ANTenna:POSition:Y
driver.receiver.antenna.position.set_y(y = 1.0)
```

Sets the antenna element position as X and Y values, relative to the receiver origin.

param y
float Range: -1e+06 to 1e+06

5.23 Repmanager

SCPI Commands :

```
REPManager:CATalog
REPManager:DELete
REPManager:EXPort
REPManager:LOAD
```

class RepmanagerCls

Repmanager commands group definition. 7 total commands, 1 Subgroups, 4 group commands

delete(*rep_name: str*, *path: str = None*, *username: str = None*, *password: str = None*) → None

```
# SCPI: REPManager:DELete
driver.repmanager.delete(rep_name = 'abc', path = 'abc', username = 'abc',␣
↪password = 'abc')
```

Deletes the entire repository from the permanent mass storage.

param rep_name
string Repository name, as configured in the workspace. If more than one repository with the same name exists, the Path must be specified.

param path
string Compete file path, as queried with the command method RsPulseSeq.Repmanager.Path.listPy. The Path must be specified, if the RepName is not unique and if Username and Passwd are used.

param username
string Required if the repository is password protected

param password
string Required if the repository is password protected

export(*rep_name: str*, *path: str*, *ps_archive_file: str*) → None

```
# SCPI: REPManager:EXPort
driver.repmanager.export(rep_name = 'abc', path = 'abc', ps_archive_file = 'abc
↪')
```

Exports the selected repository file to an archive file.

param rep_name
string Repository name, as configured in the workspace.

param path
string Compete file path, as queried with the command method RsPulseSeq.Repmanager.Path.listPy.

param ps_archive_file
Complete file path, incl. file name, and extension (*.psarch) .

get_catalog() → List[str]

```
# SCPI: REPManager:CATalog
value: List[str] = driver.repmanager.get_catalog()
```

Queries available repository elements in the database.

return
catalog: 'RepositryName','path' RepositryName is the name of the repository as defined with the command method RsPulseSeq.Repository.create Path is the compete file path

load(*rep_name: str*, *path: str = None*, *username: str = None*, *password: str = None*) → None

```
# SCPI: REPManager:LOAD
driver.repmanager.load(rep_name = 'abc', path = 'abc', username = 'abc',␣
↪password = 'abc')
```

Loads the selected repository to the workspace. If more than one repository with the same name exist, loaded is the first repository with a name match. To query the available repository elements in the database, use the command method RsPulseSeq.Repository.catalog.

param rep_name
string Repository name, as configured in the workspace.

param path
string Compete file path, as queried with the command method RsPulseSeq.Repmanager.Path.listPy. The Path must be specified, if Username and Passwd are used.

param username
string Required if the repository is password protected

param password
string Required if the repository is password protected

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.repmanager.clone()
```

Subgroups

5.23.1 Path

SCPI Commands :

```
REPManager:PATH:ADD
REPManager:PATH:DELete
REPManager:PATH:LIST
```

class PathCls

Path commands group definition. 3 total commands, 0 Subgroups, 3 group commands

delete(*delete: str*) → None

```
# SCPI: REPManager:PATH:DELete
driver.repmanager.path.delete(delete = 'abc')
```

Removes the selected file path.

param delete
string File path

get_list_py() → str

```
# SCPI: REPManager:PATH:LIST
value: str = driver.repmanager.path.get_list_py()
```

Queries the directory in that the repository files are stored.

return
list_py: string Compete file path

set_add(*add: str*) → None

```
# SCPI: REPManager:PATH:ADD
driver.repmanager.path.set_add(add = 'abc')
```

Add the selected directory.

param add
string Complete file path

5.24 Repository

SCPI Commands :

```
REPository:COMPlexity
REPository:ACCess
REPository:AUTHor
REPository:CATalog
REPository:COMMent
REPository:CREate
REPository:DATE
REPository:FILename
REPository:PATH
REPository:REMove
REPository:SAVE
REPository:SECurity
REPository:SELect
REPository:UUID
REPository:VERSion
```

class RepositoryCls

Repository commands group definition. 17 total commands, 2 Subgroups, 15 group commands

get_access() → str

```
# SCPI: REPository:ACCess
value: str = driver.repository.get_access()
```

Queries information on the access rights of the current user.

return
access: permission,login,pass,Uname permission Permission of the current user, for example RW (read-write) login,pass Login/Pass=No: Password not required Login/Pass=Yes: Password required Uname User name of the current user

get_author() → str

```
# SCPI: REPository:AUTHor
value: str = driver.repository.get_author()
```

Enters information on the author.

return
author: string

get_catalog() → str

```
# SCPI: REPository:CATalog
value: str = driver.repository.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: REPository:COMMent
value: str = driver.repository.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_complexity() → Complexity

```
# SCPI: REPository:COMPlexity
value: enums.Complexity = driver.repository.get_complexity()
```

Sets the complexity level.

return
complexity: PTRain| EMITter| DIRection

get_date() → str

```
# SCPI: REPository:DATE
value: str = driver.repository.get_date()
```

Queries the creation data.

return
date: string

get_filename() → str

```
# SCPI: REPository:FILename
value: str = driver.repository.get_filename()
```

Queries the file name of the repository archive.

return
filename: string File path, incl. file name, and extension

get_path() → str

```
# SCPI: REPository:PATH
value: str = driver.repository.get_path()
```

Queries the directory in that the repository archive is stored.

return
path: string

get_security() → SecurityLevel

```
# SCPI: REPository:SECurity
value: enums.SecurityLevel = driver.repository.get_security()
```

Sets the security level.

return
security: LEV0| LEV1| LEV2| LEV3| LEV4

get_select() → str

```
# SCPI: REPository:SELect
value: str = driver.repository.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

get_uuid() → str

```
# SCPI: REPository:UUID
value: str = driver.repository.get_uuid()
```

Queries the repository's Universally Unique Identifier (UUID) .

return
uuid: string

get_version() → str

```
# SCPI: REPository:VERSion
value: str = driver.repository.get_version()
```

Sets the repository version.

return
version: string

save() → None

```
# SCPI: REPository:SAVE
driver.repository.save()
```

Stores the repository archive. To query the storage location, use the command method RsPulseSeq.Repository.path.

save_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: REPository:SAVE
driver.repository.save_with_opc()
```

Stores the repository archive. To query the storage location, use the command method RsPulseSeq.Repository.path.

Same as save, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_author(*author: str*) → None

```
# SCPI: REPository:AUTHor
driver.repository.set_author(author = 'abc')
```

Enters information on the author.

param author
string

set_comment(*comment: str*) → None

```
# SCPI: REPository:COMMent
driver.repository.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_complexity(*complexity: Complexity*) → None

```
# SCPI: REPository:COMPlexity
driver.repository.set_complexity(complexity = enums.Complexity.DIRection)
```

Sets the complexity level.

param complexity
PTRain| EMITter| DIRection

set_create(*create: str*) → None

```
# SCPI: REPository:CREate
driver.repository.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_date(*date: str*) → None

```
# SCPI: REPository:DATE
driver.repository.set_date(date = 'abc')
```

Queries the creation data.

param date
string

set_remove(*remove: str*) → None

```
# SCPI: REPository:REMove
driver.repository.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_security(*security: SecurityLevel*) → None

```
# SCPI: REPository:SECurity
driver.repository.set_security(security = enums.SecurityLevel.LEV0)
```

Sets the security level.

param security
LEV0| LEV1| LEV2| LEV3| LEV4

set_select(*select: str*) → None

```
# SCPI: REPository:SELect
driver.repository.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_version(*version: str*) → None

```
# SCPI: REPository:VERSion
driver.repository.set_version(version = 'abc')
```

Sets the repository version.

param version
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.repository.clone()
```

Subgroups

5.24.1 Volatile

SCPI Command :

```
REPository:VOLatile:PATH
```

class VolatileCls

Volatile commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_path() → str

```
# SCPI: REPository:VOLatile:PATH
value: str = driver.repository.volatile.get_path()
```

Queries the directory in that the volatile data is saved.

return
path: string

5.24.2 Xpol

SCPI Command :

```
REPository:XPOL:ATTenuation
```

class XpolCls

Xpol commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_attenuation() → float

```
# SCPI: REPository:XPOL:ATTenuation
value: float = driver.repository.xpol.get_attenuation()
```

Sets the attenuation used to calculate the cross-polarized antenna patterns.

return
attenuation: float

set_attenuation(*attenuation: float*) → None

```
# SCPI: REPository:XPOL:ATTenuation
driver.repository.xpol.set_attenuation(attenuation = 1.0)
```

Sets the attenuation used to calculate the cross-polarized antenna patterns.

param attenuation
float

5.25 Scan

SCPI Commands :

```
SCAN:CATalog
SCAN:COMMent
SCAN:CREate
SCAN:NAME
SCAN:REMove
SCAN:SELect
SCAN:STEering
SCAN:TYPE
```

class ScanCls

Scan commands group definition. 89 total commands, 10 Subgroups, 8 group commands

get_catalog() → str

```
# SCPI: SCAN:CATalog
value: str = driver.scan.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: SCAN:COMMent
value: str = driver.scan.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_name() → str

```
# SCPI: SCAN:NAME
value: str = driver.scan.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: SCAN:SELect
value: str = driver.scan.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

get_steering() → bool

```
# SCPI: SCAN:STEering
value: bool = driver.scan.get_steering()
```

Defines whether electronic steering is used. Electronic steering is only available for scan types that use phased array antennas.

return
steering: ON| OFF| 1| 0

get_type_py() → ScanType

```
# SCPI: SCAN:TYPE
value: enums.ScanType = driver.scan.get_type_py()
```

Sets the scan type.

return
type_py: CIRCular| SECTor| RASTer| CONical| HELical| SPIRal| LSW| SIN| CUSTom| LISSajous

set_comment(*comment: str*) → None

```
# SCPI: SCAN:COMMent
driver.scan.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: SCAN:CREate
driver.scan.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: SCAN:NAME
driver.scan.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: SCAN:REMove
driver.scan.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: SCAN:SELect
driver.scan.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_steering(*steering: bool*) → None

```
# SCPI: SCAN:STEering
driver.scan.set_steering(steering = False)
```

Defines whether electronic steering is used. Electronic steering is only available for scan types that use phased array antennas.

param steering
ON| OFF| 1| 0

set_type_py(*type_py: ScanType*) → None

```
# SCPI: SCAN:TYPE
driver.scan.set_type_py(type_py = enums.ScanType.CIRCular)
```

Sets the scan type.

param type_py
CIRCular| SECTor| RASTer| CONical| HELical| SPIRal| LSW| SIN| CUSTom| LISSajous

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scan.clone()
```

Subgroups

5.25.1 Circular

SCPI Commands :

```
SCAN:CIRCular:MODE
SCAN:CIRCular:NELevation
SCAN:CIRCular:NODDing
SCAN:CIRCular:NRATe
SCAN:CIRCular:PALMer
SCAN:CIRCular:PERiod
SCAN:CIRCular:PRATe
SCAN:CIRCular:PSQuint
SCAN:CIRCular:ROTation
SCAN:CIRCular:RPM
```

class CircularCls

Circular commands group definition. 10 total commands, 0 Subgroups, 10 group commands

get_mode() → CircularMode

```
# SCPI: SCAN:CIRCular:MODE
value: enums.CircularMode = driver.scan.circular.get_mode()
```

Sets if the scan turning speed is set as a scans rate or as a period.

return
mode: RPM| SEC RPM Scan rate, set with the command method RsPulseSeq.Scan.Circular.rpm. SEC Scan period, set with the command method RsPulseSeq.Scan.Circular.period.

get_nelevation() → float

```
# SCPI: SCAN:CIRCular:NELevation
value: float = driver.scan.circular.get_nelevation()
```

Sets the elevation angle.

return
nelevation: float Range: 0.01 to 90

get_nodding() → bool

```
# SCPI: SCAN:CIRCular:NODDing
value: bool = driver.scan.circular.get_nodding()
```

Enables superimposing a horizontal nodding on the scan.

return
nodding: ON| OFF| 1| 0

get_nrate() → float

```
# SCPI: SCAN:CIRCular:NRATe
value: float = driver.scan.circular.get_nrate()
```

Sets the elevation rate.

return
nrate: float Range: 0.01 to 2000

get_palmer() → bool

```
# SCPI: SCAN:CIRCular:PALMer
value: bool = driver.scan.circular.get_palmer()
```

Enables superimposing a conical scan on the current scan.

return
palmer: ON| OFF| 1| 0

get_period() → float

```
# SCPI: SCAN:CIRCular:PERiod
value: float = driver.scan.circular.get_period()
```

Sets the time it takes for the antenna to turn once.

return
period: float Range: 0.006 to 6000

get_prate() → float

```
# SCPI: SCAN:CIRCular:PRATe
value: float = driver.scan.circular.get_prate()
```

Sets the scan rate.

return
prate: float Range: 0.1 to 1000

get_psquint() → float

```
# SCPI: SCAN:CIRCular:PSQuint
value: float = driver.scan.circular.get_psquint()
```

Sets the squint angle.

return
psquint: float Range: 0.05 to 45

get_rotation() → Rotation

```
# SCPI: SCAN:CIRCular:ROTation
value: enums.Rotation = driver.scan.circular.get_rotation()
```

Sets the rotation direction of the antenna.

return
rotation: CW| CCW

get_rpm() → float

```
# SCPI: SCAN:CIRCular:RPM
value: float = driver.scan.circular.get_rpm()
```

Sets the rotation speed of the antenna.

return
rpm: float Range: 0.01 to 1000, Unit: degree/s

set_mode(*mode: CircularMode*) → None

```
# SCPI: SCAN:CIRCular:MODE
driver.scan.circular.set_mode(mode = enums.CircularMode.RPM)
```

Sets if the scan turning speed is set as a scans rate or as a period.

param mode
RPM| SEC RPM Scan rate, set with the command method RsPulseSeq.Scan.Circular.rpm. SEC Scan period, set with the command method RsPulseSeq.Scan.Circular.period.

set_nelevation(*nelevation: float*) → None

```
# SCPI: SCAN:CIRCular:NELevation
driver.scan.circular.set_nelevation(nelevation = 1.0)
```

Sets the elevation angle.

param nelevation
float Range: 0.01 to 90

set_nodding(*nodding: bool*) → None

```
# SCPI: SCAN:CIRCular:NODDing
driver.scan.circular.set_nodding(nodding = False)
```

Enables superimposing a horizontal nodding on the scan.

param nodding
ON| OFF| 1| 0

set_nrate(*nrate: float*) → None

```
# SCPI: SCAN:CIRCular:NRATe
driver.scan.circular.set_nrate(nrate = 1.0)
```

Sets the elevation rate.

param nrate
float Range: 0.01 to 2000

set_palmer(*palmer: bool*) → None

```
# SCPI: SCAN:CIRCular:PALMer
driver.scan.circular.set_palmer(palmer = False)
```

Enables superimposing a conical scan on the current scan.

param palmer
ON| OFF| 1| 0

set_period(*period: float*) → None

```
# SCPI: SCAN:CIRCular:PERiod
driver.scan.circular.set_period(period = 1.0)
```

Sets the time it takes for the antenna to turn once.

param period
float Range: 0.006 to 6000

set_prate(*prate: float*) → None

```
# SCPI: SCAN:CIRCular:PRATe
driver.scan.circular.set_prate(prate = 1.0)
```

Sets the scan rate.

param prate
float Range: 0.1 to 1000

set_psquint(*psquint: float*) → None

```
# SCPI: SCAN:CIRCular:PSQuint
driver.scan.circular.set_psquint(psquint = 1.0)
```

Sets the squint angle.

param psquint
float Range: 0.05 to 45

set_rotation(*rotation: Rotation*) → None

```
# SCPI: SCAN:CIRCular:ROTation
driver.scan.circular.set_rotation(rotation = enums.Rotation.CCW)
```

Sets the rotation direction of the antenna.

param rotation
CW| CCW

set_rpm(*rpm: float*) → None

```
# SCPI: SCAN:CIRCular:RPM
driver.scan.circular.set_rpm(rpm = 1.0)
```

Sets the rotation speed of the antenna.

param rpm
float Range: 0.01 to 1000, Unit: degree/s

5.25.2 Conical

SCPI Commands :

```
SCAN:CONical:RATE
SCAN:CONical:ROTation
SCAN:CONical:SQUint
```

class ConicalCls

Conical commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_rate() → float

```
# SCPI: SCAN:CONical:RATE
value: float = driver.scan.conical.get_rate()
```

Sets the turning speed.

return
rate: float Range: 0.01 to 100000, Unit: degree/s

get_rotation() → Rotation

```
# SCPI: SCAN:CONical:ROTation
value: enums.Rotation = driver.scan.conical.get_rotation()
```

Sets the rotation direction of the antenna.

return
rotation: CW| CCW

get_squint() → float

```
# SCPI: SCAN:CONical:SQUint
value: float = driver.scan.conical.get_squint()
```

Sets the offset angle of the antenna beam, that means for the conical antenna the parameter sets the radius of the scanned circle.

return
squint: float Range: 0.05 to 15, Unit: degree

set_rate(*rate: float*) → None

```
# SCPI: SCAN:CONical:RATE
driver.scan.conical.set_rate(rate = 1.0)
```

Sets the turning speed.

param rate
float Range: 0.01 to 100000, Unit: degree/s

set_rotation(*rotation: Rotation*) → None

```
# SCPI: SCAN:CONical:ROTation
driver.scan.conical.set_rotation(rotation = enums.Rotation.CCW)
```

Sets the rotation direction of the antenna.

param rotation
CW| CCW

set_squint(*squint: float*) → None

```
# SCPI: SCAN:CONical:SQUint
driver.scan.conical.set_squint(squint = 1.0)
```

Sets the offset angle of the antenna beam, that means for the conical antenna the parameter sets the radius of the scanned circle.

param squint
float Range: 0.05 to 15, Unit: degree

5.25.3 Custom

class CustomCls

Custom commands group definition. 13 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scan.custom.clone()
```

Subgroups

5.25.3.1 Entry

SCPI Commands :

```
SCAN:CUSTom:ENTRy:AZIMuth
SCAN:CUSTom:ENTRy:CLEar
SCAN:CUSTom:ENTRy:COUNt
SCAN:CUSTom:ENTRy:DELete
SCAN:CUSTom:ENTRy:DWELl
```

(continues on next page)

(continued from previous page)

```
SCAN:CUSTom:ENTRy:ELEVation
SCAN:CUSTom:ENTRy:INSert
SCAN:CUSTom:ENTRy:JUMPtype
SCAN:CUSTom:ENTRy:SELect
SCAN:CUSTom:ENTRy:TRANstime
```

class EntryCls

Entry commands group definition. 11 total commands, 1 Subgroups, 10 group commands

clear() → None

```
# SCPI: SCAN:CUSTom:ENTRy:CLEar
driver.scan.custom.entry.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:CLEar
driver.scan.custom.entry.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:DELete
driver.scan.custom.entry.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_azimuth() → float

```
# SCPI: SCAN:CUSTom:ENTRy:AZIMuth
value: float = driver.scan.custom.entry.get_azimuth()
```

Sets the azimuth of the scan position.

return
azimuth: float Range: -180 to 180

get_count() → float

```
# SCPI: SCAN:CUSTom:ENTRy:COUNt
value: float = driver.scan.custom.entry.get_count()
```

Queries the number of existing items.

return
count: integer

get_dwell() → float

```
# SCPI: SCAN:CUSTom:ENTRy:DWELl
value: float = driver.scan.custom.entry.get_dwell()
```

Sets how long the scan stays in a position.

return
dwell: float Range: 0 to 3600

get_elevation() → float

```
# SCPI: SCAN:CUSTom:ENTRy:ELEVation
value: float = driver.scan.custom.entry.get_elevation()
```

Sets the elevation of the scan position.

return
elevation: float Range: -90 to 90

get_jump_type() → bool

```
# SCPI: SCAN:CUSTom:ENTRy:JUMPtype
value: bool = driver.scan.custom.entry.get_jump_type()
```

Defines how to move to the next position, either with a jump or with a transition. For transitions, you need to define a transition time.

return
jump_type: ON| OFF| 1| 0 ON | 1 Jump enabled. OFF | 0 Transition enabled.

get_select() → float

```
# SCPI: SCAN:CUSTom:ENTRy:SELect
value: float = driver.scan.custom.entry.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_trans_time() → float

```
# SCPI: SCAN:CUSTom:ENTRy:TRANstime
value: float = driver.scan.custom.entry.get_trans_time()
```

Sets the time for the transition between two positions.

return
trans_time: float Range: 0 to 3600

set_azimuth(*azimuth: float*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:AZIMuth
driver.scan.custom.entry.set_azimuth(azimuth = 1.0)
```

Sets the azimuth of the scan position.

param azimuth
float Range: -180 to 180

set_dwell(*dwell: float*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:DWELl
driver.scan.custom.entry.set_dwell(dwell = 1.0)
```

Sets how long the scan stays in a position.

param dwell
float Range: 0 to 3600

set_elevation(*elevation: float*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:ELEVation
driver.scan.custom.entry.set_elevation(elevation = 1.0)
```

Sets the elevation of the scan position.

param elevation
float Range: -90 to 90

set_insert(*insert: float*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:INSert
driver.scan.custom.entry.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_jump_type(*jump_type: bool*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:JUMPtype
driver.scan.custom.entry.set_jump_type(jump_type = False)
```

Defines how to move to the next position, either with a jump or with a transition. For transitions, you need to define a transition time.

param jump_type
ON| OFF| 1| 0 ON | 1 Jump enabled. OFF | 0 Transition enabled.

set_select(*select: float*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:SELect
driver.scan.custom.entry.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_trans_time(*trans_time: float*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:TRANstime
driver.scan.custom.entry.set_trans_time(trans_time = 1.0)
```

Sets the time for the transition between two positions.

param trans_time
float Range: 0 to 3600

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scan.custom.entry.clone()
```

Subgroups

5.25.3.1.1 Add

SCPI Command :

```
SCAN:CUSTom:ENTRy:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCAN:CUSTom:ENTRy:ADD
driver.scan.custom.entry.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCAN:CUSTom:ENTRy:ADD
driver.scan.custom.entry.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.25.3.2 ImportPy

SCPI Command :

```
SCAN:CUSTom:IMPort:FILE
```

class ImportPyCls

ImportPy commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_file() → str

```
# SCPI: SCAN:CUSTom:IMPort:FILE
value: str = driver.scan.custom.importPy.get_file()
```

Sets the file to import.

return
file: string

set_file(*file: str*) → None

```
# SCPI: SCAN:CUSTom:IMPort:FILE
driver.scan.custom.importPy.set_file(file = 'abc')
```

Sets the file to import.

param file
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scan.custom.importPy.clone()
```

Subgroups

5.25.3.2.1 Exec

SCPI Command :

```
SCAN:CUSTom:IMPort:EXEC
```

class ExecCls

Exec commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCAN:CUSTom:IMPort:EXEC
driver.scan.custom.importPy.exec.set()
```

Starts importings the file.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCAN:CUSTom:IMPort:EXEC
driver.scan.custom.importPy.exec.set_with_opc()
```

Starts importings the file.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.25.4 Helical

SCPI Commands :

```
SCAN:HELical:RETRace
SCAN:HELical:ROTation
SCAN:HELical:RPM
SCAN:HELical:TURNs
```

class HelicalCls

Helical commands group definition. 5 total commands, 1 Subgroups, 4 group commands

get_retrace() → float

```
# SCPI: SCAN:HELical:RETRace
value: float = driver.scan.helical.get_retrace()
```

Sets the speed for the antenna to return to the initial orientation.

return
retrace: float Range: 0 to 1

get_rotation() → Rotation

```
# SCPI: SCAN:HELical:ROTation
value: enums.Rotation = driver.scan.helical.get_rotation()
```

Sets the rotation direction of the antenna.

return
rotation: CW| CCW

get_rpm() → float

```
# SCPI: SCAN:HELical:RPM
value: float = driver.scan.helical.get_rpm()
```

Sets the rotation speed of the antenna.

return
rpm: float Range: 0.01 to 1000, Unit: degree/s

get_turns() → float

```
# SCPI: SCAN:HELical:TURNs
value: float = driver.scan.helical.get_turns()
```

Sets the number of turns.

return
turns: float Range: 1 to 30

set_retrace(*retrace: float*) → None

```
# SCPI: SCAN:HELical:RETRace
driver.scan.helical.set_retrace(retrace = 1.0)
```

Sets the speed for the antenna to return to the initial orientation.

param retrace
float Range: 0 to 1

set_rotation(*rotation: Rotation*) → None

```
# SCPI: SCAN:HELical:ROTation
driver.scan.helical.set_rotation(rotation = enums.Rotation.CCW)
```

Sets the rotation direction of the antenna.

param rotation
CW| CCW

set_rpm(*rpm: float*) → None

```
# SCPI: SCAN:HELical:RPM
driver.scan.helical.set_rpm(rpm = 1.0)
```

Sets the rotation speed of the antenna.

param rpm
float Range: 0.01 to 1000, Unit: degree/s

set_turns(*turns: float*) → None

```
# SCPI: SCAN:HELical:TURNs
driver.scan.helical.set_turns(turns = 1.0)
```

Sets the number of turns.

param turns
float Range: 1 to 30

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scan.helical.clone()
```

Subgroups

5.25.4.1 Elevation

SCPI Command :

```
SCAN:HELical:ELEVation:STEP
```

class ElevationCls

Elevation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_step() → float

```
# SCPI: SCAN:HELical:ELEVation:STEP
value: float = driver.scan.helical.elevation.get_step()
```

Sets the step width with that the antenna changes its elevation.

return
step: float Range: 0.01 to 11.25, Unit: degree

set_step(*step: float*) → None

```
# SCPI: SCAN:HELical:ELEVation:STEP
driver.scan.helical.elevation.set_step(step = 1.0)
```

Sets the step width with that the antenna changes its elevation.

param step
float Range: 0.01 to 11.25, Unit: degree

5.25.5 Lissajous

SCPI Commands :

```
SCAN:LISSajous:AMPX
SCAN:LISSajous:AMPZ
SCAN:LISSajous:FREQ
SCAN:LISSajous:PHIX
SCAN:LISSajous:PHIZ
SCAN:LISSajous:XFACtor
SCAN:LISSajous:ZFACtor
```

class LissajousCls

Lissajous commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_am_px() → float

```
# SCPI: SCAN:LISSajous:AMPX
value: float = driver.scan.lissajous.get_am_px()
```

Sets the magnitudes of two harmonic vibrations.

return
am_px: No help available

get_am_pz() → float

```
# SCPI: SCAN:LISSajous:AMPZ
value: float = driver.scan.lissajous.get_am_pz()
```

Sets the magnitudes of two harmonic vibrations.

return
am_pz: float Range: 0.01 to 45

get_freq() → float

```
# SCPI: SCAN:LISSajous:FREQ
value: float = driver.scan.lissajous.get_freq()
```

Sets the base frequency.

return
freq: float Range: 0.01 to 1000

get_phix() → float

```
# SCPI: SCAN:LISSajous:PHIX
value: float = driver.scan.lissajous.get_phix()
```

Sets the phases of the two harmonic vibrations.

return
phix: No help available

get_phiz() → float

```
# SCPI: SCAN:LISSajous:PHIZ
value: float = driver.scan.lissajous.get_phiz()
```

Sets the phases of the two harmonic vibrations.

return
phiz: float Range: 0 to 360

get_xfactor() → float

```
# SCPI: SCAN:LISSajous:XFACtor
value: float = driver.scan.lissajous.get_xfactor()
```

Sets the ratio between the two angular frequencies.

return
xfactor: No help available

get_zfactor() → float

```
# SCPI: SCAN:LISSajous:ZFACtor
value: float = driver.scan.lissajous.get_zfactor()
```

Sets the ratio between the two angular frequencies.

return
zfactor: float Range: 1 to 10

set_am_px(*am_px: float*) → None

```
# SCPI: SCAN:LISSajous:AMPX
driver.scan.lissajous.set_am_px(am_px = 1.0)
```

Sets the magnitudes of two harmonic vibrations.

param am_px
float Range: 0.01 to 45

set_am_pz(*am_pz: float*) → None

```
# SCPI: SCAN:LISSajous:AMPZ
driver.scan.lissajous.set_am_pz(am_pz = 1.0)
```

Sets the magnitudes of two harmonic vibrations.

param am_pz
float Range: 0.01 to 45

set_freq(*freq: float*) → None

```
# SCPI: SCAN:LISSajous:FREQ
driver.scan.lissajous.set_freq(freq = 1.0)
```

Sets the base frequency.

param freq
float Range: 0.01 to 1000

set_phix(*phix: float*) → None

```
# SCPI: SCAN:LISSajous:PHIX
driver.scan.lissajous.set_phix(phix = 1.0)
```

Sets the phases of the two harmonic vibrations.

param phix
float Range: 0 to 360

set_phiz(*phiz: float*) → None

```
# SCPI: SCAN:LISSajous:PHIZ
driver.scan.lissajous.set_phiz(phiz = 1.0)
```

Sets the phases of the two harmonic vibrations.

param phiz
float Range: 0 to 360

set_xfactor(*xfactor: float*) → None

```
# SCPI: SCAN:LISSajous:XFACtor
driver.scan.lissajous.set_xfactor(xfactor = 1.0)
```

Sets the ratio between the two angular frequencies.

param xfactor
float Range: 1 to 10

set_zfactor(*zfactor: float*) → None

```
# SCPI: SCAN:LISSajous:ZFACtor
driver.scan.lissajous.set_zfactor(zfactor = 1.0)
```

Sets the ratio between the two angular frequencies.

param zfactor
float Range: 1 to 10

5.25.6 Lsw

SCPI Commands :

```
SCAN:LSW:DIRection
SCAN:LSW:DWELl
SCAN:LSW:LOBes
SCAN:LSW:ROTation
SCAN:LSW:SQUint
```

class LswCls

Lsw commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_direction() → LswDirection

```
# SCPI: SCAN:LSW:DIRection
value: enums.LswDirection = driver.scan.lsw.get_direction()
```

Sets the horizontal or vertical switching direction.

return
direction: H| V

get_dwell() → float

```
# SCPI: SCAN:LSW:DWELl
value: float = driver.scan.lsw.get_dwell()
```

Sets the speed with that the antenna switches between the lobes.

return
dwell: float Range: 1e-06 to 1

get_lobes() → LobesCount

```
# SCPI: SCAN:LSW:LOBes
value: enums.LobesCount = driver.scan.lsw.get_lobes()
```

Set the number of lobes.

return
lobes: 2| 4

get_rotation() → Rotation

```
# SCPI: SCAN:LSW:ROTation
value: enums.Rotation = driver.scan.lsw.get_rotation()
```

Sets the rotation direction of the antenna.

return
rotation: CW| CCW

get_squint() → float

```
# SCPI: SCAN:LSW:SQUint
value: float = driver.scan.lsw.get_squint()
```

Sets the offset angle of the antenna beam, that means for the conical antenna the parameter sets the radius of the scanned circle.

return
squint: float Range: 0.05 to 15, Unit: degree

set_direction(*direction: LswDirection*) → None

```
# SCPI: SCAN:LSW:DIRection
driver.scan.lsw.set_direction(direction = enums.LswDirection.H)
```

Sets the horizontal or vertical switching direction.

param direction
H| V

set_dwell(*dwell: float*) → None

```
# SCPI: SCAN:LSW:DWELl
driver.scan.lsw.set_dwell(dwell = 1.0)
```

Sets the speed with that the antenna switches between the lobes.

param dwell
float Range: 1e-06 to 1

set_lobes(*lobes: LobesCount*) → None

```
# SCPI: SCAN:LSW:LOBes
driver.scan.lsw.set_lobes(lobes = enums.LobesCount._2)
```

Set the number of lobes.

param lobes
2| 4

set_rotation(*rotation: Rotation*) → None

```
# SCPI: SCAN:LSW:ROTation
driver.scan.lsw.set_rotation(rotation = enums.Rotation.CCW)
```

Sets the rotation direction of the antenna.

param rotation
CW| CCW

set_squint(*squint: float*) → None

```
# SCPI: SCAN:LSW:SQUint
driver.scan.lsw.set_squint(squint = 1.0)
```

Sets the offset angle of the antenna beam, that means for the conical antenna the parameter sets the radius of the scanned circle.

param squint
float Range: 0.05 to 15, Unit: degree

5.25.7 Raster

SCPI Commands :

```
SCAN:RASTer:BARS
SCAN:RASTer:BARTranstime
SCAN:RASTer:BARWidth
SCAN:RASTer:DIRection
SCAN:RASTer:FLYBack
SCAN:RASTer:PALMer
SCAN:RASTer:PRATe
SCAN:RASTer:PSQuint
SCAN:RASTer:RATE
SCAN:RASTer:RETRace
SCAN:RASTer:REWind
SCAN:RASTer:UNIDirection
SCAN:RASTer:WIDTh
```

class RasterCls

Raster commands group definition. 13 total commands, 0 Subgroups, 13 group commands

get_bar_trans_time() → float

```
# SCPI: SCAN:RASTer:BARTranstime
value: float = driver.scan.raster.get_bar_trans_time()
```

Transition time between two bars in bidirectional scan mode.

return
bar_trans_time: float Range: 0 to 1, Unit: seconds

get_bar_width() → float

```
# SCPI: SCAN:RASTer:BARWidth
value: float = driver.scan.raster.get_bar_width()
```

Sets the distance between two consecutive scanned bars (sectors) .

return
bar_width: float Range: 0.1 to 9, Unit: m

get_bars() → float

```
# SCPI: SCAN:RASTer:BARS
value: float = driver.scan.raster.get_bars()
```

Sets the number of scanned bars (sectors) .

return
bars: float Range: 1 to 30

get_direction() → RasterDirection

```
# SCPI: SCAN:RASTer:DIRection
value: enums.RasterDirection = driver.scan.raster.get_direction()
```

Sets the scanning direction.

return
direction: HORizontal| VERTical

get_flyback() → float

```
# SCPI: SCAN:RASTer:FLYBack
value: float = driver.scan.raster.get_flyback()
```

Sets the Flyback time for the antenna working in unidirectional mode.

return
flyback: float Range: 0 to 1, Unit: s

get_palmer() → bool

```
# SCPI: SCAN:RASTer:PALMer
value: bool = driver.scan.raster.get_palmer()
```

Enables superimposing a conical scan on the current scan.

return
palmer: ON| OFF| 1| 0

get_prate() → float

```
# SCPI: SCAN:RASTer:PRATe
value: float = driver.scan.raster.get_prate()
```

Sets the scan rate.

return
prate: float Range: 0.1 to 1000

get_psquint() → float

```
# SCPI: SCAN:RASTer:PSQuint
value: float = driver.scan.raster.get_psquint()
```

Sets the squint angle.

return
psquint: float Range: 0.05 to 45

get_rate() → float

```
# SCPI: SCAN:RASTer:RATE
value: float = driver.scan.raster.get_rate()
```

Sets the turning speed.

return
rate: float Range: 0.01 to 100000, Unit: degree/s

get_retrace() → float

```
# SCPI: SCAN:RASTer:RETRace
value: float = driver.scan.raster.get_retrace()
```

Sets the speed for the antenna to return to the initial orientation.

return
retrace: float Range: 0 to 1

get_rewind() → bool

```
# SCPI: SCAN:RASTer:REWind
value: bool = driver.scan.raster.get_rewind()
```

If enabled, the antenna scans forwards and backwards.

return
rewind: ON| OFF| 1| 0

get_uni_direction() → bool

```
# SCPI: SCAN:RASTer:UNIDirection
value: bool = driver.scan.raster.get_uni_direction()
```

Enables a unidirectional scan mode.

return
uni_direction: ON| OFF| 1| 0

get_width() → float

```
# SCPI: SCAN:RASTer:WIDTh
value: float = driver.scan.raster.get_width()
```

Sets the width of the sector to be scanned.

return
width: float Range: 0.1 to 360, Unit: degree

set_bar_trans_time(*bar_trans_time: float*) → None

```
# SCPI: SCAN:RASTer:BARTranstime
driver.scan.raster.set_bar_trans_time(bar_trans_time = 1.0)
```

Transition time between two bars in bidirectional scan mode.

param bar_trans_time
float Range: 0 to 1, Unit: seconds

set_bar_width(*bar_width: float*) → None

```
# SCPI: SCAN:RASTer:BARWidth
driver.scan.raster.set_bar_width(bar_width = 1.0)
```

Sets the distance between two consecutive scanned bars (sectors) .

param bar_width
float Range: 0.1 to 9, Unit: m

set_bars(*bars: float*) → None

```
# SCPI: SCAN:RASTer:BARS
driver.scan.raster.set_bars(bars = 1.0)
```

Sets the number of scanned bars (sectors) .

param bars
float Range: 1 to 30

set_direction(*direction: RasterDirection*) → None

```
# SCPI: SCAN:RASTer:DIRection
driver.scan.raster.set_direction(direction = enums.RasterDirection.HORizontal)
```

Sets the scanning direction.

param direction
HORizontal| VERTical

set_flyback(*flyback: float*) → None

```
# SCPI: SCAN:RASTer:FLYBack
driver.scan.raster.set_flyback(flyback = 1.0)
```

Sets the Flyback time for the antenna working in unidirectional mode.

param flyback
float Range: 0 to 1, Unit: s

set_palmer(*palmer: bool*) → None

```
# SCPI: SCAN:RASTer:PALMer
driver.scan.raster.set_palmer(palmer = False)
```

Enables superimposing a conical scan on the current scan.

param palmer
ON| OFF| 1| 0

set_prate(*prate: float*) → None

```
# SCPI: SCAN:RASTer:PRATe
driver.scan.raster.set_prate(prate = 1.0)
```

Sets the scan rate.

param prate
float Range: 0.1 to 1000

set_psquint(*psquint: float*) → None

```
# SCPI: SCAN:RASTer:PSQuint
driver.scan.raster.set_psquint(psquint = 1.0)
```

Sets the squint angle.

param psquint
float Range: 0.05 to 45

set_rate(*rate: float*) → None

```
# SCPI: SCAN:RASTer:RATE
driver.scan.raster.set_rate(rate = 1.0)
```

Sets the turning speed.

param rate
float Range: 0.01 to 100000, Unit: degree/s

set_retrace(*retrace: float*) → None

```
# SCPI: SCAN:RASTer:RETRace
driver.scan.raster.set_retrace(retrace = 1.0)
```

Sets the speed for the antenna to return to the initial orientation.

param retrace
float Range: 0 to 1

set_rewind(*rewind: bool*) → None

```
# SCPI: SCAN:RASTer:REWind
driver.scan.raster.set_rewind(rewind = False)
```

If enabled, the antenna scans forwards and backwards.

param rewind
ON| OFF| 1| 0

set_uni_direction(*uni_direction: bool*) → None

```
# SCPI: SCAN:RASTer:UNIDirection
driver.scan.raster.set_uni_direction(uni_direction = False)
```

Enables a unidirectional scan mode.

param uni_direction
ON| OFF| 1| 0

set_width(*width: float*) → None

```
# SCPI: SCAN:RASTer:WIDTh
driver.scan.raster.set_width(width = 1.0)
```

Sets the width of the sector to be scanned.

param width
float Range: 0.1 to 360, Unit: degree

5.25.8 Sector

SCPI Commands :

```
SCAN:SECTor:FLYBack
SCAN:SECTor:NELevation
SCAN:SECTor:NODDing
SCAN:SECTor:NRATe
SCAN:SECTor:PALMer
SCAN:SECTor:PRATe
SCAN:SECTor:PSQuint
SCAN:SECTor:RATE
```

(continues on next page)

(continued from previous page)

```
SCAN:SECTor:UNIDirection
SCAN:SECTor:WIDTh
```

class SectorCls

Sector commands group definition. 10 total commands, 0 Subgroups, 10 group commands

get_flyback() → float

```
# SCPI: SCAN:SECTor:FLYBack
value: float = driver.scan.sector.get_flyback()
```

Sets the Flyback time for the antenna working in unidirectional mode.

return
flyback: float Range: 0 to 1, Unit: s

get_nelevation() → float

```
# SCPI: SCAN:SECTor:NELevation
value: float = driver.scan.sector.get_nelevation()
```

Sets the elevation angle.

return
nelevation: float Range: 0.01 to 90

get_nodding() → bool

```
# SCPI: SCAN:SECTor:NODDing
value: bool = driver.scan.sector.get_nodding()
```

Enables superimposing a horizontal nodding on the scan.

return
nodding: ON| OFF| 1| 0

get_nrate() → float

```
# SCPI: SCAN:SECTor:NRATe
value: float = driver.scan.sector.get_nrate()
```

Sets the elevation rate.

return
nrate: float Range: 0.01 to 2000

get_palmer() → bool

```
# SCPI: SCAN:SECTor:PALMer
value: bool = driver.scan.sector.get_palmer()
```

Enables superimposing a conical scan on the current scan.

return
palmer: ON| OFF| 1| 0

get_prate() → float

```
# SCPI: SCAN:SECTor:PRATe
value: float = driver.scan.sector.get_prate()
```

Sets the scan rate.

return
prate: float Range: 0.1 to 1000

get_psquint() → float

```
# SCPI: SCAN:SECTor:PSQuint
value: float = driver.scan.sector.get_psquint()
```

Sets the squint angle.

return
psquint: float Range: 0.05 to 45

get_rate() → float

```
# SCPI: SCAN:SECTor:RATE
value: float = driver.scan.sector.get_rate()
```

Sets the turning speed.

return
rate: float Range: 0.01 to 100000, Unit: degree/s

get_uni_direction() → bool

```
# SCPI: SCAN:SECTor:UNIDirection
value: bool = driver.scan.sector.get_uni_direction()
```

Enables a unidirectional scan mode.

return
uni_direction: ON| OFF| 1| 0

get_width() → float

```
# SCPI: SCAN:SECTor:WIDTh
value: float = driver.scan.sector.get_width()
```

Sets the width of the sector to be scanned.

return
width: float Range: 0.1 to 360, Unit: degree

set_flyback(*flyback: float*) → None

```
# SCPI: SCAN:SECTor:FLYBack
driver.scan.sector.set_flyback(flyback = 1.0)
```

Sets the Flyback time for the antenna working in unidirectional mode.

param flyback
float Range: 0 to 1, Unit: s

set_nelevation(*nelevation: float*) → None

```
# SCPI: SCAN:SECTor:NELevation
driver.scan.sector.set_nelevation(nelevation = 1.0)
```

Sets the elevation angle.

param nelevation
float Range: 0.01 to 90

set_nodding(*nodding: bool*) → None

```
# SCPI: SCAN:SECTor:NODDing
driver.scan.sector.set_nodding(nodding = False)
```

Enables superimposing a horizontal nodding on the scan.

param nodding
ON| OFF| 1| 0

set_nrate(*nrate: float*) → None

```
# SCPI: SCAN:SECTor:NRATe
driver.scan.sector.set_nrate(nrate = 1.0)
```

Sets the elevation rate.

param nrate
float Range: 0.01 to 2000

set_palmer(*palmer: bool*) → None

```
# SCPI: SCAN:SECTor:PALMer
driver.scan.sector.set_palmer(palmer = False)
```

Enables superimposing a conical scan on the current scan.

param palmer
ON| OFF| 1| 0

set_prate(*prate: float*) → None

```
# SCPI: SCAN:SECTor:PRATe
driver.scan.sector.set_prate(prate = 1.0)
```

Sets the scan rate.

param prate
float Range: 0.1 to 1000

set_psquint(*psquint: float*) → None

```
# SCPI: SCAN:SECTor:PSQuint
driver.scan.sector.set_psquint(psquint = 1.0)
```

Sets the squint angle.

param psquint
float Range: 0.05 to 45

set_rate(*rate: float*) → None

```
# SCPI: SCAN:SECTor:RATE
driver.scan.sector.set_rate(rate = 1.0)
```

Sets the turning speed.

param rate
float Range: 0.01 to 100000, Unit: degree/s

set_uni_direction(*uni_direction: bool*) → None

```
# SCPI: SCAN:SECTor:UNIDirection
driver.scan.sector.set_uni_direction(uni_direction = False)
```

Enables a unidirectional scan mode.

param uni_direction
ON| OFF| 1| 0

set_width(*width: float*) → None

```
# SCPI: SCAN:SECTor:WIDTh
driver.scan.sector.set_width(width = 1.0)
```

Sets the width of the sector to be scanned.

param width
float Range: 0.1 to 360, Unit: degree

5.25.9 Sin

SCPI Commands :

```
SCAN:SIN:HEIGht
SCAN:SIN:INVersion
SCAN:SIN:RATE
SCAN:SIN:ROTation
SCAN:SIN:UNIDirection
SCAN:SIN:WIDTh
```

class SinCls

Sin commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_height() → float

```
# SCPI: SCAN:SIN:HEIGht
value: float = driver.scan.sin.get_height()
```

Sets the amplitude of the sine wave.

return
height: float Range: 1 to 90

get_inversion() → bool

```
# SCPI: SCAN:SIN:INVersion
value: bool = driver.scan.sin.get_inversion()
```

Sets whether the upper or the down (mirrored) sine wave is used first.

return
inversion: ON| OFF| 1| 0 OFF Upper sine first ON Down sine first

get_rate() → float

```
# SCPI: SCAN:SIN:RATE
value: float = driver.scan.sin.get_rate()
```

Sets the turning speed.

return
rate: float Range: 0.01 to 100000, Unit: degree/s

get_rotation() → Rotation

```
# SCPI: SCAN:SIN:ROTation
value: enums.Rotation = driver.scan.sin.get_rotation()
```

Sets the rotation direction of the antenna.

return
rotation: CW| CCW

get_uni_direction() → bool

```
# SCPI: SCAN:SIN:UNIDirection
value: bool = driver.scan.sin.get_uni_direction()
```

Enables a unidirectional scan mode.

return
uni_direction: ON| OFF| 1| 0

get_width() → float

```
# SCPI: SCAN:SIN:WIDTh
value: float = driver.scan.sin.get_width()
```

Sets the angle on the XY plane between the origin and the end of the scan.

return
width: float Range: 1 to 180

set_height(*height: float*) → None

```
# SCPI: SCAN:SIN:HEIGht
driver.scan.sin.set_height(height = 1.0)
```

Sets the amplitude of the sine wave.

param height
float Range: 1 to 90

set_inversion(*inversion: bool*) → None

```
# SCPI: SCAN:SIN:INVersion
driver.scan.sin.set_inversion(inversion = False)
```

Sets whether the upper or the down (mirrored) sine wave is used first.

param inversion
ON| OFF| 1| 0 OFF Upper sine first ON Down sine first

set_rate(*rate: float*) → None

```
# SCPI: SCAN:SIN:RATE
driver.scan.sin.set_rate(rate = 1.0)
```

Sets the turning speed.

param rate
float Range: 0.01 to 100000, Unit: degree/s

set_rotation(*rotation: Rotation*) → None

```
# SCPI: SCAN:SIN:ROTation
driver.scan.sin.set_rotation(rotation = enums.Rotation.CCW)
```

Sets the rotation direction of the antenna.

param rotation
CW| CCW

set_uni_direction(*uni_direction: bool*) → None

```
# SCPI: SCAN:SIN:UNIDirection
driver.scan.sin.set_uni_direction(uni_direction = False)
```

Enables a unidirectional scan mode.

param uni_direction
ON| OFF| 1| 0

set_width(*width: float*) → None

```
# SCPI: SCAN:SIN:WIDTh
driver.scan.sin.set_width(width = 1.0)
```

Sets the angle on the XY plane between the origin and the end of the scan.

param width
float Range: 1 to 180

5.25.10 Spiral

SCPI Commands :

```
SCAN:SPIRal:PALMer
SCAN:SPIRal:PRATe
SCAN:SPIRal:PSQuint
SCAN:SPIRal:RETRace
SCAN:SPIRal:ROTation
SCAN:SPIRal:ROUNds
SCAN:SPIRal:RTIMe
SCAN:SPIRal:STEP
SCAN:SPIRal:UNIDirection
```

class SpiralCls

Spiral commands group definition. 9 total commands, 0 Subgroups, 9 group commands

get_palmer() → bool

```
# SCPI: SCAN:SPIRal:PALMer
value: bool = driver.scan.spiral.get_palmer()
```

Enables superimposing a conical scan on the current scan.

return
palmer: ON| OFF| 1| 0

get_prate() → float

```
# SCPI: SCAN:SPIRal:PRATe
value: float = driver.scan.spiral.get_prate()
```

Sets the scan rate.

return
prate: float Range: 0.1 to 1000

get_psquint() → float

```
# SCPI: SCAN:SPIRal:PSQuint
value: float = driver.scan.spiral.get_psquint()
```

Sets the squint angle.

return
psquint: float Range: 0.05 to 45

get_retrace() → float

```
# SCPI: SCAN:SPIRal:RETRace
value: float = driver.scan.spiral.get_retrace()
```

Sets the speed for the antenna to return to the initial orientation.

return
retrace: float Range: 0 to 1

get_rotation() → Rotation

```
# SCPI: SCAN:SPIRal:ROTation
value: enums.Rotation = driver.scan.spiral.get_rotation()
```

Sets the rotation direction of the antenna.

return
rotation: CW| CCW

get_rounds() → float

```
# SCPI: SCAN:SPIRal:ROUNds
value: float = driver.scan.spiral.get_rounds()
```

Sets the number of rounds the antenna performs.

return
rounds: float Range: 0.1 to 15

get_rtime() → float

```
# SCPI: SCAN:SPIRal:RTIMe
value: float = driver.scan.spiral.get_rtime()
```

Sets the turning speed of the antenna.

return
rtime: float Range: 0.01 to 10, Unit: degree/s

get_step() → float

```
# SCPI: SCAN:SPIRal:STEP
value: float = driver.scan.spiral.get_step()
```

Determines the step size to increase the scan radius.

return
step: float Range: 1 to 11.25, Unit: degree

get_uni_direction() → bool

```
# SCPI: SCAN:SPIRal:UNIDirection
value: bool = driver.scan.spiral.get_uni_direction()
```

Enables a unidirectional scan mode.

return
uni_direction: ON| OFF| 1| 0

set_palmer(*palmer: bool*) → None

```
# SCPI: SCAN:SPIRal:PALMer
driver.scan.spiral.set_palmer(palmer = False)
```

Enables superimposing a conical scan on the current scan.

param palmer
ON| OFF| 1| 0

set_prate(*prate: float*) → None

```
# SCPI: SCAN:SPIRal:PRATe
driver.scan.spiral.set_prate(prate = 1.0)
```

Sets the scan rate.

param prate
float Range: 0.1 to 1000

set_psquint(*psquint: float*) → None

```
# SCPI: SCAN:SPIRal:PSQuint
driver.scan.spiral.set_psquint(psquint = 1.0)
```

Sets the squint angle.

param psquint
float Range: 0.05 to 45

set_retrace(*retrace: float*) → None

```
# SCPI: SCAN:SPIRal:RETRace
driver.scan.spiral.set_retrace(retrace = 1.0)
```

Sets the speed for the antenna to return to the initial orientation.

param retrace
float Range: 0 to 1

set_rotation(*rotation: Rotation*) → None

```
# SCPI: SCAN:SPIRal:ROTation
driver.scan.spiral.set_rotation(rotation = enums.Rotation.CCW)
```

Sets the rotation direction of the antenna.

param rotation
CW| CCW

set_rounds(*rounds: float*) → None

```
# SCPI: SCAN:SPIRal:ROUNds
driver.scan.spiral.set_rounds(rounds = 1.0)
```

Sets the number of rounds the antenna performs.

param rounds
float Range: 0.1 to 15

set_rtime(*rtime: float*) → None

```
# SCPI: SCAN:SPIRal:RTIMe
driver.scan.spiral.set_rtime(rtime = 1.0)
```

Sets the turning speed of the antenna.

param rtime
float Range: 0.01 to 10, Unit: degree/s

set_step(*step: float*) → None

```
# SCPI: SCAN:SPIRal:STEP
driver.scan.spiral.set_step(step = 1.0)
```

Determines the step size to increase the scan radius.

param step
float Range: 1 to 11.25, Unit: degree

set_uni_direction(*uni_direction: bool*) → None

```
# SCPI: SCAN:SPIRal:UNIDirection
driver.scan.spiral.set_uni_direction(uni_direction = False)
```

Enables a unidirectional scan mode.

param uni_direction
ON| OFF| 1| 0

5.26 Scenario

SCPI Commands :

```
SCENario:CATalog
SCENario:COMMent
SCENario:CREate
SCENario:ID
SCENario:NAME
SCENario:REMove
SCENario:SANitize
SCENario:SELect
SCENario:STARt
SCENario:STATe
SCENario:STOP
SCENario:TYPE
SCENario:WAVeform
```

class ScenarioCls

Scenario commands group definition. 461 total commands, 17 Subgroups, 13 group commands

get_catalog() → str

```
# SCPI: SCENario:CATalog
value: str = driver.scenario.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: SCENario:COMMent
value: str = driver.scenario.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_id() → float

```
# SCPI: SCENario:ID
value: float = driver.scenario.get_id()
```

Queries the database identifier of the selected scenario.

return
idn: float

get_name() → str

```
# SCPI: SCENario:NAME
value: str = driver.scenario.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: SCENario:SELect
value: str = driver.scenario.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

get_state() → State

```
# SCPI: SCENario:STATe
value: enums.State = driver.scenario.get_state()
```

Queries the current scenario status.

return
state: IDLE| RUN

get_type_py() → ScenarioType

```
# SCPI: SCENario:TYPE
value: enums.ScenarioType = driver.scenario.get_type_py()
```

Sets the scenario type.

return
type_py: SEQuence| CSEQuence| EMITter| CEMitter| LOCalized| DF| PDW | WAVeform

get_waveform() → str

```
# SCPI: SCENario:WAVeform
value: str = driver.scenario.get_waveform()
```

Specify the name of the 'Signal Generation' output file.

return
waveform: string

set_comment(*comment: str*) → None

```
# SCPI: SCENario:COMMent
driver.scenario.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: SCENario:CREate
driver.scenario.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: SCENario:NAME
driver.scenario.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: SCENario:REMove
driver.scenario.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_sanitize(*sanitize: SanitizeScenario*) → None

```
# SCPI: SCENario:SANitize
driver.scenario.set_sanitize(sanitize = enums.SanitizeScenario.ALL)
```

Removes uploaded waveforms from the hard disk of the signal generator.

param sanitize
SCENario| REPository| ALL SCENario Removes the current scenario waveforms REPository Removes the waveforms of all scenarios from the current repository ALL Removes all waveforms created by the R&S Pulse Sequencer

set_select(*select: str*) → None

```
# SCPI: SCENario:SELect
driver.scenario.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_type_py(*type_py: ScenarioType*) → None

```
# SCPI: SCENario:TYPE
driver.scenario.set_type_py(type_py = enums.ScenarioType.CEMitter)
```

Sets the scenario type.

param type_py
SEQuence| CSEQuence| EMITter| CEMitter| LOCalized| DF| PDW | WAVeform

set_waveform(*waveform: str*) → None

```
# SCPI: SCENario:WAVeform
driver.scenario.set_waveform(waveform = 'abc')
```

Specify the name of the 'Signal Generation' output file.

param waveform
string

start() → None

```
# SCPI: SCENario:STARt
driver.scenario.start()
```

Starts the signal generation.

start_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:STARt
driver.scenario.start_with_opc()
```

Starts the signal generation.

Same as start, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

stop() → None

```
# SCPI: SCENario:STOP
driver.scenario.stop()
```

Stops the signal calculation.

stop_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:STOP
driver.scenario.stop_with_opc()
```

Stops the signal calculation.

Same as stop, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.clone()
```

Subgroups

5.26.1 Cache

class CacheCls

Cache commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cache.clone()
```

Subgroups

5.26.1.1 Repository

SCPI Commands :

```
SCENario:CACHe:REPository:CLEar
SCENario:CACHe:REPository:VALid
```

class RepositoryCls

Repository commands group definition. 4 total commands, 1 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:CACHe:REPository:CLEar
driver.scenario.cache.repository.clear()
```

Deletes the files from the volatile/repository memory.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CACHe:REPository:CLEar
driver.scenario.cache.repository.clear_with_opc()
```

Deletes the files from the volatile/repository memory.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_valid() → bool

```
# SCPI: SCENario:CACHe:REPository:VALid
value: bool = driver.scenario.cache.repository.get_valid()
```

Queries whether the volatile/repository memory contains a valid signal file.

return
valid: ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cache.repository.clone()
```

Subgroups

5.26.1.1.1 Enable

SCPI Commands :

```
SCENario:CACHe:REPository:ENABle:INTerleave
SCENario:CACHe:REPository:ENABle
```

class EnableCls

Enable commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_interleave() → bool

```
# SCPI: SCENario:CACHe:REPository:ENABle:INTerleave
value: bool = driver.scenario.cache.repository.enable.get_interleave()
```

Enables file storage in the repository when interleaving is active.

return
interleave: ON| OFF| 1| 0

get_value() → bool

```
# SCPI: SCENario:CACHe:REPository:ENABle
value: bool = driver.scenario.cache.repository.enable.get_value()
```

Enables file storage in the repository.

return
enable: ON| OFF| 1| 0

set_interleave(*interleave: bool*) → None

```
# SCPI: SCENario:CACHe:REPository:ENABle:INTerleave
driver.scenario.cache.repository.enable.set_interleave(interleave = False)
```

Enables file storage in the repository when interleaving is active.

param interleave
ON| OFF| 1| 0

set_value(*enable: bool*) → None

```
# SCPI: SCENario:CACHe:REPository:ENABle
driver.scenario.cache.repository.enable.set_value(enable = False)
```

Enables file storage in the repository.

param enable
ON| OFF| 1| 0

5.26.1.2 Volatile

SCPI Commands :

```
SCENario:CACHe:VOLatile:CLEar
SCENario:CACHe:VOLatile:VALid
```

class VolatileCls

Volatile commands group definition. 4 total commands, 2 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:CACHe:VOLatile:CLEar
driver.scenario.cache.volatile.clear()
```

Deletes the files from the volatile/repository memory.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CACHe:VOLatile:CLEar
driver.scenario.cache.volatile.clear_with_opc()
```

Deletes the files from the volatile/repository memory.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_valid() → bool

```
# SCPI: SCENario:CACHe:VOLatile:VALid
value: bool = driver.scenario.cache.volatile.get_valid()
```

Queries whether the volatile/repository memory contains a valid signal file.

return
valid: ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cache.volatile.clone()
```

Subgroups

5.26.1.2.1 Release

SCPI Command :

```
SCENario:CACHe:VOLatile:RELease
```

class ReleaseCls

Release commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CACHe:VOLatile:RELease
driver.scenario.cache.volatile.release.set()
```

Exports and stores the generated signal files in the repository.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CACHe:VOLatile:RELease
driver.scenario.cache.volatile.release.set_with_opc()
```

Exports and stores the generated signal files in the repository.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.1.2.2 Restore

SCPI Command :

```
SCENario:CACHe:VOLatile:RESTore
```

class RestoreCls

Restore commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CACHe:VOLatile:RESTore
driver.scenario.cache.volatile.restore.set()
```

Loads signal files from the storage.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CACHe:VOLatile:RESTore
driver.scenario.cache.volatile.restore.set_with_opc()
```

Loads signal files from the storage.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.2 Calculate

SCPI Command :

```
SCENario:CALCulate
```

class CalculateCls

Calculate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CALCulate
driver.scenario.calculate.set()
```

Starts the signal calculation.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CALCulate
driver.scenario.calculate.set_with_opc()
```

Starts the signal calculation.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.3 Cemit

SCPI Commands :

```
SCENario:CEMit:ALIas
SCENario:CEMit:CLEar
SCENario:CEMit:CURRent
SCENario:CEMit:DELete
SCENario:CEMit:ENABle
SCENario:CEMit:FQOFfset
SCENario:CEMit:FREQuency
SCENario:CEMit:LDELay
SCENario:CEMit:LEVel
SCENario:CEMit:LVABs
SCENario:CEMit:PRIority
SCENario:CEMit:SCNDelay
```

(continues on next page)

(continued from previous page)

```
SCENario:CEMit:SELect
SCENario:CEMit:THReshold
```

class CemitCls

Cemit commands group definition. 52 total commands, 7 Subgroups, 14 group commands

clear() → None

```
# SCPI: SCENario:CEMit:CLEar
driver.scenario.cemit.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CEMit:CLEar
driver.scenario.cemit.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:CEMit:DELete
driver.scenario.cemit.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:CEMit:ALIas
value: str = driver.scenario.cemit.get_alias()
```

Enters an alias name.

return
alias: string

get_current() → float

```
# SCPI: SCENario:CEMit:CURRent
value: float = driver.scenario.cemit.get_current()
```

No command help available

return
current: No help available

get_enable() → bool

```
# SCPI: SCENario:CEMit:ENABle
value: bool = driver.scenario.cemit.get_enable()
```

If enabled, the PDW list is included in the output file.

return
enable: ON| OFF| 1| 0

get_fq_qffset() → float

```
# SCPI: SCENario:CEMit:FQOFfset
value: float = driver.scenario.cemit.get_fq_qffset()
```

Sets the frequency offset for the selected emitter.

return
fq_offset: float Range: -2e+09 to 2e+09

get_frequency() → float

```
# SCPI: SCENario:CEMit:FREQuency
value: float = driver.scenario.cemit.get_frequency()
```

Sets the frequency for the selected emitter.

return
frequency: No help available

get_ldelay() → float

```
# SCPI: SCENario:CEMit:LDELay
value: float = driver.scenario.cemit.get_ldelay()
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

return
ldelay: float Range: -1e+09 to 1e+09

get_level() → float

```
# SCPI: SCENario:CEMit:LEVel
value: float = driver.scenario.cemit.get_level()
```

Adds a level offset.

return
level: float Range: -200 to 0

get_lvabs() → float

```
# SCPI: SCENario:CEMit:LVABs
value: float = driver.scenario.cemit.get_lvabs()
```

Sets the absolute level for the selected PDW list.

return
lvabs: float Range: -130 to 30

get_priority() → float

```
# SCPI: SCENario:CEMit:PRIority
value: float = driver.scenario.cemit.get_priority()
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

return
priority: float Range: 1 to 100

get_scn_delay() → float

```
# SCPI: SCENario:CEMit:SCNDelay
value: float = driver.scenario.cemit.get_scn_delay()
```

Sets the scan delay for the selected emitter.

return
scn_delay: float Range: -3600 to 3600

get_select() → float

```
# SCPI: SCENario:CEMit:SELect
value: float = driver.scenario.cemit.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_threshold() → float

```
# SCPI: SCENario:CEMit:THReshold
value: float = driver.scenario.cemit.get_threshold()
```

Sets a threshold. Pulses at levels below this threshold are omitted.

return
threshold: float Range: -100 to 0

set_alias(*alias: str*) → None

```
# SCPI: SCENario:CEMit:ALIas
driver.scenario.cemit.set_alias(alias = 'abc')
```

Enters an alias name.

param alias
string

set_current(*current: float*) → None

```
# SCPI: SCENario:CEMit:CURRent
driver.scenario.cemit.set_current(current = 1.0)
```

No command help available

param current
No help available

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:CEMit:ENABle
driver.scenario.cemit.set_enable(enable = False)
```

If enabled, the PDW list is included in the output file.

param enable
ON| OFF| 1| 0

set_fq_qffset(*fq_offset: float*) → None

```
# SCPI: SCENario:CEMit:FQOFfset
driver.scenario.cemit.set_fq_qffset(fq_offset = 1.0)
```

Sets the frequency offset for the selected emitter.

param fq_offset
float Range: -2e+09 to 2e+09

set_ldelay(*ldelay: float*) → None

```
# SCPI: SCENario:CEMit:LDELay
driver.scenario.cemit.set_ldelay(ldelay = 1.0)
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

param ldelay
float Range: -1e+09 to 1e+09

set_level(*level: float*) → None

```
# SCPI: SCENario:CEMit:LEVel
driver.scenario.cemit.set_level(level = 1.0)
```

Adds a level offset.

param level
float Range: -200 to 0

set_lvabs(*lvabs: float*) → None

```
# SCPI: SCENario:CEMit:LVABs
driver.scenario.cemit.set_lvabs(lvabs = 1.0)
```

Sets the absolute level for the selected PDW list.

param lvabs
float Range: -130 to 30

set_priority(*priority: float*) → None

```
# SCPI: SCENario:CEMit:PRIority
driver.scenario.cemit.set_priority(priority = 1.0)
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

param priority
float Range: 1 to 100

set_scn_delay(*scn_delay: float*) → None

```
# SCPI: SCENario:CEMit:SCNDelay
driver.scenario.cemit.set_scn_delay(scn_delay = 1.0)
```

Sets the scan delay for the selected emitter.

param scn_delay
float Range: -3600 to 3600

set_select(*select: float*) → None

```
# SCPI: SCENario:CEMit:SELect
driver.scenario.cemit.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_threshold(*threshold: float*) → None

```
# SCPI: SCENario:CEMit:THReshold
driver.scenario.cemit.set_threshold(threshold = 1.0)
```

Sets a threshold. Pulses at levels below this threshold are omitted.

param threshold
float Range: -100 to 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cemit.clone()
```

Subgroups

5.26.3.1 Add

SCPI Command :

```
SCENario:CEMit:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CEMit:ADD
driver.scenario.cemit.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CEMit:ADD
driver.scenario.cemit.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.3.2 Direction

SCPI Commands :

```
SCENario:CEMit:DIRection:PITCh
SCENario:CEMit:DIRection:ROLL
SCENario:CEMit:DIRection:YAW
```

class DirectionCls

Direction commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_pitch() → float

```
# SCPI: SCENario:CEMit:DIRection:PITCh
value: float = driver.scenario.cemit.direction.get_pitch()
```

Sets the pitch.

return
pitch: float Range: -90 to 90, Unit: grad

get_roll() → float

```
# SCPI: SCENario:CEMit:DIRection:ROLL
value: float = driver.scenario.cemit.direction.get_roll()
```

Sets the roll.

return
roll: float Range: 0 to 360

get_yaw() → float

```
# SCPI: SCENario:CEMit:DIRection:YAW
value: float = driver.scenario.cemit.direction.get_yaw()
```

Sets the yaw.

return
yaw: float Range: 0 to 360

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:CEMit:DIRection:PITCh
driver.scenario.cemit.direction.set_pitch(pitch = 1.0)
```

Sets the pitch.

param pitch
float Range: -90 to 90, Unit: grad

set_roll(*roll: float*) → None

```
# SCPI: SCENario:CEMit:DIRection:ROLL
driver.scenario.cemit.direction.set_roll(roll = 1.0)
```

Sets the roll.

param roll
float Range: 0 to 360

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:CEMit:DIRection:YAW
driver.scenario.cemit.direction.set_yaw(yaw = 1.0)
```

Sets the yaw.

param yaw
float Range: 0 to 360

5.26.3.3 Emitter

SCPI Commands :

```
SCENario:CEMit:EMITter:ENABle
SCENario:CEMit:EMITter
```

class EmitterCls

Emitter commands group definition. 5 total commands, 1 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SCENario:CEMit:EMITter:ENABle
value: bool = driver.scenario.cemit.emitter.get_enable()
```

In a map-based sceanrio, enable selected item for calculation.

return
enable: ON| OFF| 1| 0

get_value() → str

```
# SCPI: SCENario:CEMit:EMITter
value: str = driver.scenario.cemit.emitter.get_value()
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

return
emitter: string

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:CEMit:EMITter:ENABle
driver.scenario.cemit.emitter.set_enable(enable = False)
```

In a map-based sceanrio, enable selected item for calculation.

param enable
ON| OFF| 1| 0

set_value(*emitter: str*) → None

```
# SCPI: SCENario:CEMit:EMITter
driver.scenario.cemit.emitter.set_value(emitter = 'abc')
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

param emitter
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cemit.emitter.clone()
```

Subgroups

5.26.3.3.1 Mode

SCPI Commands :

```
SCENario:CEMit:EMITter:MODE:BEAM
SCENario:CEMit:EMITter:MODE:TRACkrec
SCENario:CEMit:EMITter:MODE
```

class ModeCls

Mode commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_beam() → float

```
# SCPI: SCENario:CEMit:EMITter:MODE:BEAM
value: float = driver.scenario.cemit.emitter.mode.get_beam()
```

Sets the used beam of the current mode.

return
beam: float Range: 1 to 32

get_track_rec() → bool

```
# SCPI: SCENario:CEMit:EMITter:MODE:TRACkrec
value: bool = driver.scenario.cemit.emitter.mode.get_track_rec()
```

If enabled, the scan follows the receiver automatically.

return
track_rec: ON| OFF| 1| 0

get_value() → float

```
# SCPI: SCENario:CEMit:EMITter:MODE
value: float = driver.scenario.cemit.emitter.mode.get_value()
```

Set the emitter mode.

return
mode: float Range: 1 to 32

set_beam(*beam: float*) → None

```
# SCPI: SCENario:CEMit:EMITter:MODE:BEAM
driver.scenario.cemit.emitter.mode.set_beam(beam = 1.0)
```

Sets the used beam of the current mode.

param beam
float Range: 1 to 32

set_track_rec(*track_rec: bool*) → None

```
# SCPI: SCENario:CEMit:EMITter:MODE:TRACkrec
driver.scenario.cemit.emitter.mode.set_track_rec(track_rec = False)
```

If enabled, the scan follows the receiver automatically.

param track_rec
ON| OFF| 1| 0

set_value(*mode: float*) → None

```
# SCPI: SCENario:CEMit:EMITter:MODE
driver.scenario.cemit.emitter.mode.set_value(mode = 1.0)
```

Set the emitter mode.

param mode
float Range: 1 to 32

5.26.3.4 Group

SCPI Commands :

```
SCENario:CEMit:GROup:ALIas
SCENario:CEMit:GROup:CATalog
SCENario:CEMit:GROup:CLEar
SCENario:CEMit:GROup:COUNt
SCENario:CEMit:GROup:DELete
SCENario:CEMit:GROup:SELect
SCENario:CEMit:GROup
```

class GroupCls

Group commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:CEMit:GROup:CLEar
driver.scenario.cemit.group.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CEMit:GROup:CLEar
driver.scenario.cemit.group.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:CEMit:GROup:DELete
driver.scenario.cemit.group.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:CEMit:GROup:ALIas
value: str = driver.scenario.cemit.group.get_alias()
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

return
alias: string

get_catalog() → str

```
# SCPI: SCENario:CEMit:GROup:CATalog
value: str = driver.scenario.cemit.group.get_catalog()
```

Queries the alias names of the configured interleaving groups.

return
catalog: string A list of coma-separated alias names.

get_count() → float

```
# SCPI: SCENario:CEMit:GROup:COUNt
value: float = driver.scenario.cemit.group.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:CEMit:GROup:SELect
value: float = driver.scenario.cemit.group.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → str

```
# SCPI: SCENario:CEMit:GROup
value: str = driver.scenario.cemit.group.get_value()
```

Assigns the emitter to one of the available interleaving groups.

return
group: string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

set_alias(*alias: str*) → None

```
# SCPI: SCENario:CEMit:GROup:ALIas
driver.scenario.cemit.group.set_alias(alias = 'abc')
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

param alias
string

set_select(*select: float*) → None

```
# SCPI: SCENario:CEMit:GROup:SELect
driver.scenario.cemit.group.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*group: str*) → None

```
# SCPI: SCENario:CEMit:GROup
driver.scenario.cemit.group.set_value(group = 'abc')
```

Assigns the emitter to one of the available interleaving groups.

param group
string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cemit.group.clone()
```

Subgroups

5.26.3.4.1 Add

SCPI Command :

```
SCENario:CEMit:GROup:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CEMit:GROup:ADD
driver.scenario.cemit.group.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CEMit:GROup:ADD
driver.scenario.cemit.group.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.3.5 Interleaving

SCPI Commands :

```
SCENario:CEMit:INTerleaving:MODE
SCENario:CEMit:INTerleaving:FREQagility
SCENario:CEMit:INTerleaving
```

class InterleavingCls

Interleaving commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_freq_agility() → bool

```
# SCPI: SCENario:CEMit:INTerleaving:FREQagility
value: bool = driver.scenario.cemit.interleaving.get_freq_agility()
```

Enables frequency agility in interleaving. Requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

return
freq_agility: ON| OFF| 1| 0

get_mode() → InterleaveMode

```
# SCPI: SCENario:CEMit:INTerleaving:MODE
value: enums.InterleaveMode = driver.scenario.cemit.interleaving.get_mode()
```

Select the mode for interleaving.

return
mode: DROP| MERGe DROP Interleaving uses a priority-based dropping algorithm. MERGE Emitters or PDW lists are merged into multiple output files using groups.

get_value() → bool

```
# SCPI: SCENario:CEMit:INTerleaving
value: bool = driver.scenario.cemit.interleaving.get_value()
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

return
interleaving: ON| OFF| 1| 0

set_freq_agility(*freq_agility: bool*) → None

```
# SCPI: SCENario:CEMit:INTerleaving:FREQagility
driver.scenario.cemit.interleaving.set_freq_agility(freq_agility = False)
```

Enables frequency agility in interleaving. Requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

param freq_agility
ON| OFF| 1| 0

set_mode(*mode: InterleaveMode*) → None

```
# SCPI: SCENario:CEMit:INTerleaving:MODE
driver.scenario.cemit.interleaving.set_mode(mode = enums.InterleaveMode.DROP)
```

Select the mode for interleaving.

param mode
DROP| MERGe DROP Interleaving uses a priority-based dropping algorithm. MERGE Emitters or PDW lists are merged into multiple output files using groups.

set_value(*interleaving: bool*) → None

```
# SCPI: SCENario:CEMit:INTerleaving
driver.scenario.cemit.interleaving.set_value(interleaving = False)
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

param interleaving
ON| OFF| 1| 0

5.26.3.6 Marker

SCPI Commands :

```
SCENario:CEMit:MARKer:AUTO
SCENario:CEMit:MARKer:FALL
SCENario:CEMit:MARKer:FORCe
SCENario:CEMit:MARKer:GATE
SCENario:CEMit:MARKer:POST
SCENario:CEMit:MARKer:PRE
SCENario:CEMit:MARKer:RISE
SCENario:CEMit:MARKer:WIDTh
```

class MarkerCls

Marker commands group definition. 10 total commands, 1 Subgroups, 8 group commands

get_auto() → float

```
# SCPI: SCENario:CEMit:MARKer:AUTO
value: float = driver.scenario.cemit.marker.get_auto()
```

Enables the marker for restart.

return
auto: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_fall() → float

```
# SCPI: SCENario:CEMit:MARKer:FALL
value: float = driver.scenario.cemit.marker.get_fall()
```

Enables the marker for fall time.

return
fall: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_force() → bool

```
# SCPI: SCENario:CEMit:MARKer:FORCe
value: bool = driver.scenario.cemit.marker.get_force()
```

Determines how the marker is handled.

return
force: ON| OFF| 1| 0 ON | 1 Forces the selected marker type for every pulse of the selected emitter OFF | 0 Leaves the marker unchanged, as defined in the pulses and sequences of this emitter.

get_gate() → float

```
# SCPI: SCENario:CEMit:MARKer:GATE
value: float = driver.scenario.cemit.marker.get_gate()
```

Enables marker for gate.

return
gate: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_post() → float

```
# SCPI: SCENario:CEMit:MARKer:POST
value: float = driver.scenario.cemit.marker.get_post()
```

Enables marker for post time.

return
post: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_pre() → float

```
# SCPI: SCENario:CEMit:MARKer:PRE
value: float = driver.scenario.cemit.marker.get_pre()
```

Enables marker for pre time.

return
pre: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_rise() → float

```
# SCPI: SCENario:CEMit:MARKer:RISE
value: float = driver.scenario.cemit.marker.get_rise()
```

Enables marker for rise time.

return
rise: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_width() → float

```
# SCPI: SCENario:CEMit:MARKer:WIDTh
value: float = driver.scenario.cemit.marker.get_width()
```

Sets marker for the pulse width.

return
width: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_auto(*auto: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:AUTO
driver.scenario.cemit.marker.set_auto(auto = 1.0)
```

Enables the marker for restart.

param auto
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_fall(*fall: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:FALL
driver.scenario.cemit.marker.set_fall(fall = 1.0)
```

Enables the marker for fall time.

param fall
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_force(*force: bool*) → None

```
# SCPI: SCENario:CEMit:MARKer:FORCe
driver.scenario.cemit.marker.set_force(force = False)
```

Determines how the marker is handled.

param force
ON| OFF| 1| 0 ON | 1 Forces the selected marker type for every pulse of the selected emitter OFF | 0 Leaves the marker unchanged, as defined in the pulses and sequences of this emitter.

set_gate(*gate: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:GATE
driver.scenario.cemit.marker.set_gate(gate = 1.0)
```

Enables marker for gate.

param gate
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_post(*post: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:POST
driver.scenario.cemit.marker.set_post(post = 1.0)
```

Enables marker for post time.

param post
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_pre(*pre: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:PRE
driver.scenario.cemit.marker.set_pre(pre = 1.0)
```

Enables marker for pre time.

param pre
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_rise(*rise: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:RISE
driver.scenario.cemit.marker.set_rise(rise = 1.0)
```

Enables marker for rise time.

param rise
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_width(*width: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:WIDTh
driver.scenario.cemit.marker.set_width(width = 1.0)
```

Sets marker for the pulse width.

param width
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cemit.marker.clone()
```

Subgroups

5.26.3.6.1 Time

SCPI Commands :

```
SCENario:CEMit:MARKer:TIME:POST
SCENario:CEMit:MARKer:TIME:PRE
```

class TimeCls

Time commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_post() → float

```
# SCPI: SCENario:CEMit:MARKer:TIME:POST
value: float = driver.scenario.cemit.marker.time.get_post()
```

Specifies post marker time.

return
post: float Range: 0 to 3600

get_pre() → float

```
# SCPI: SCENario:CEMit:MARKer:TIME:PRE
value: float = driver.scenario.cemit.marker.time.get_pre()
```

Specifies pre marker time.

return
pre: float Range: 0 to 3600

set_post(*post: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:TIME:POST
driver.scenario.cemit.marker.time.set_post(post = 1.0)
```

Specifies post marker time.

param post
float Range: 0 to 3600

set_pre(*pre: float*) → None

```
# SCPI: SCENario:CEMit:MARKer:TIME:PRE
driver.scenario.cemit.marker.time.set_pre(pre = 1.0)
```

Specifies pre marker time.

param pre
float Range: 0 to 3600

5.26.3.7 Mchg

SCPI Commands :

```
SCENario:CEMit:MCHG:CLEar
SCENario:CEMit:MCHG:COUNt
SCENario:CEMit:MCHG:DELete
SCENario:CEMit:MCHG:SELect
SCENario:CEMit:MCHG:STARt
SCENario:CEMit:MCHG:STATe
SCENario:CEMit:MCHG:STOP
```

class MchgCls

Mchg commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:CEMit:MCHG:CLEar
driver.scenario.cemit.mchg.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CEMit:MCHG:CLEar
driver.scenario.cemit.mchg.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:CEMit:MCHG:DELete
driver.scenario.cemit.mchg.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SCENario:CEMit:MCHG:COUNt
value: float = driver.scenario.cemit.mchg.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:CEMit:MCHG:SELect
value: float = driver.scenario.cemit.mchg.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_start() → float

```
# SCPI: SCENario:CEMit:MCHG:STARt
value: float = driver.scenario.cemit.mchg.get_start()
```

Sets the start and end time per mode entry.

return
start: No help available

get_state() → bool

```
# SCPI: SCENario:CEMit:MCHG:STATe
value: bool = driver.scenario.cemit.mchg.get_state()
```

Enables mode changes.

return
state: ON| OFF| 1| 0

get_stop() → float

```
# SCPI: SCENario:CEMit:MCHG:STOP
value: float = driver.scenario.cemit.mchg.get_stop()
```

Sets the start and end time per mode entry.

return
stop: float

set_select(*select: float*) → None

```
# SCPI: SCENario:CEMit:MCHG:SELect
driver.scenario.cemit.mchg.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_start(*start: float*) → None

```
# SCPI: SCENario:CEMit:MCHG:STARt
driver.scenario.cemit.mchg.set_start(start = 1.0)
```

Sets the start and end time per mode entry.

param start
float

set_state(*state: bool*) → None

```
# SCPI: SCENario:CEMit:MCHG:STATe
driver.scenario.cemit.mchg.set_state(state = False)
```

Enables mode changes.

param state
ON| OFF| 1| 0

set_stop(*stop: float*) → None

```
# SCPI: SCENario:CEMit:MCHG:STOP
driver.scenario.cemit.mchg.set_stop(stop = 1.0)
```

Sets the start and end time per mode entry.

param stop
float

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cemit.mchg.clone()
```

Subgroups

5.26.3.7.1 Add

SCPI Command :

```
SCENario:CEMit:MCHG:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CEMit:MCHG:ADD
driver.scenario.cemit.mchg.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CEMit:MCHG:ADD
driver.scenario.cemit.mchg.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.4 Cpdw

SCPI Commands :

```
SCENario:CPDW:ALIas
SCENario:CPDW:CLEar
SCENario:CPDW:DELete
SCENario:CPDW:ENABle
SCENario:CPDW:FREQ
SCENario:CPDW:INTerleaving
SCENario:CPDW:LDELay
SCENario:CPDW:LEVel
SCENario:CPDW:LVABs
SCENario:CPDW:NAME
SCENario:CPDW:PRIority
```

(continues on next page)

(continued from previous page)

```
SCENario:CPDW:SELect
SCENario:CPDW:THReshold
```

class CpdwCls

Cpdw commands group definition. 22 total commands, 2 Subgroups, 13 group commands

clear() → None

```
# SCPI: SCENario:CPDW:CLEar
driver.scenario.cpdw.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CPDW:CLEar
driver.scenario.cpdw.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:CPDW:DELete
driver.scenario.cpdw.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:CPDW:ALIas
value: str = driver.scenario.cpdw.get_alias()
```

Enters an alias name.

return
alias: string

get_enable() → bool

```
# SCPI: SCENario:CPDW:ENABle
value: bool = driver.scenario.cpdw.get_enable()
```

If enabled, the PDW list is included in the output file.

return
enable: ON| OFF| 1| 0

get_freq() → float

```
# SCPI: SCENario:CPDW:FREQ
value: float = driver.scenario.cpdw.get_freq()
```

Sets the frequency for the selected emitter.

return
freq: float Range: -1000 to 1e+11

get_interleaving() → bool

```
# SCPI: SCENario:CPDW:INTerleaving
value: bool = driver.scenario.cpdw.get_interleaving()
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

return
interleaving: ON| OFF| 1| 0

get_ldelay() → float

```
# SCPI: SCENario:CPDW:LDELay
value: float = driver.scenario.cpdw.get_ldelay()
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

return
ldelay: float Range: -1e+09 to 1e+09

get_level() → float

```
# SCPI: SCENario:CPDW:LEVel
value: float = driver.scenario.cpdw.get_level()
```

Adds a level offset.

return
level: float Range: -200 to 0

get_lvabs() → float

```
# SCPI: SCENario:CPDW:LVABs
value: float = driver.scenario.cpdw.get_lvabs()
```

Sets the absolute level for the selected PDW list.

return
lvabs: float Range: -130 to 30

get_name() → str

```
# SCPI: SCENario:CPDW:NAME
value: str = driver.scenario.cpdw.get_name()
```

Selects the waveform element, used to import the PDW list. Query the list of waveform elements with the command method RsPulseSeq.Waveform.catalog.

return
name: string

get_priority() → float

```
# SCPI: SCENario:CPDW:PRIority
value: float = driver.scenario.cpdw.get_priority()
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

return
priority: float Range: 1 to 100

get_select() → float

```
# SCPI: SCENario:CPDW:SELect
value: float = driver.scenario.cpdw.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_threshold() → float

```
# SCPI: SCENario:CPDW:THReshold
value: float = driver.scenario.cpdw.get_threshold()
```

Sets a threshold. Pulses at levels below this threshold are omitted.

return
threshold: float Range: -100 to 0

set_alias(*alias: str*) → None

```
# SCPI: SCENario:CPDW:ALIas
driver.scenario.cpdw.set_alias(alias = 'abc')
```

Enters an alias name.

param alias
string

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:CPDW:ENABle
driver.scenario.cpdw.set_enable(enable = False)
```

If enabled, the PDW list is included in the output file.

param enable
ON| OFF| 1| 0

set_freq(*freq: float*) → None

```
# SCPI: SCENario:CPDW:FREQ
driver.scenario.cpdw.set_freq(freq = 1.0)
```

Sets the frequency for the selected emitter.

param freq
float Range: -1000 to 1e+11

set_interleaving(*interleaving: bool*) → None

```
# SCPI: SCENario:CPDW:INTerleaving
driver.scenario.cpdw.set_interleaving(interleaving = False)
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

param interleaving
ON| OFF| 1| 0

set_ldelay(*ldelay: float*) → None

```
# SCPI: SCENario:CPDW:LDELay
driver.scenario.cpdw.set_ldelay(ldelay = 1.0)
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

param ldelay
float Range: -1e+09 to 1e+09

set_level(*level: float*) → None

```
# SCPI: SCENario:CPDW:LEVel
driver.scenario.cpdw.set_level(level = 1.0)
```

Adds a level offset.

param level
float Range: -200 to 0

set_lvabs(*lvabs: float*) → None

```
# SCPI: SCENario:CPDW:LVABs
driver.scenario.cpdw.set_lvabs(lvabs = 1.0)
```

Sets the absolute level for the selected PDW list.

param lvabs
float Range: -130 to 30

set_name(*name: str*) → None

```
# SCPI: SCENario:CPDW:NAME
driver.scenario.cpdw.set_name(name = 'abc')
```

Selects the waveform element, used to import the PDW list. Query the list of waveform elements with the command method RsPulseSeq.Waveform.catalog.

param name
string

set_priority(*priority: float*) → None

```
# SCPI: SCENario:CPDW:PRIority
driver.scenario.cpdw.set_priority(priority = 1.0)
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

param priority
float Range: 1 to 100

set_select(*select: float*) → None

```
# SCPI: SCENario:CPDW:SELect
driver.scenario.cpdw.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_threshold(*threshold: float*) → None

```
# SCPI: SCENario:CPDW:THReshold
driver.scenario.cpdw.set_threshold(threshold = 1.0)
```

Sets a threshold. Pulses at levels below this threshold are omitted.

param threshold
float Range: -100 to 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cpdw.clone()
```

Subgroups

5.26.4.1 Add

SCPI Command :

```
SCENario:CPDW:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CPDW:ADD
driver.scenario.cpdw.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CPDW:ADD
driver.scenario.cpdw.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.4.2 Group

SCPI Commands :

```
SCENario:CPDW:GROup:ALIas
SCENario:CPDW:GROup:CATalog
SCENario:CPDW:GROup:CLEar
SCENario:CPDW:GROup:COUNt
SCENario:CPDW:GROup:DELete
SCENario:CPDW:GROup:SELect
SCENario:CPDW:GROup
```

class GroupCls

Group commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:CPDW:GROup:CLEar
driver.scenario.cpdw.group.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CPDW:GROup:CLEar
driver.scenario.cpdw.group.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:CPDW:GROup:DELete
driver.scenario.cpdw.group.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:CPDW:GROup:ALIas
value: str = driver.scenario.cpdw.group.get_alias()
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

return
alias: string

get_catalog() → str

```
# SCPI: SCENario:CPDW:GROup:CATalog
value: str = driver.scenario.cpdw.group.get_catalog()
```

Queries the alias names of the configured interleaving groups.

return
catalog: string A list of coma-separated alias names.

get_count() → float

```
# SCPI: SCENario:CPDW:GROup:COUNt
value: float = driver.scenario.cpdw.group.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:CPDW:GROup:SELect
value: float = driver.scenario.cpdw.group.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → str

```
# SCPI: SCENario:CPDW:GROup
value: str = driver.scenario.cpdw.group.get_value()
```

Assigns the emitter to one of the available interleaving groups.

return
group: string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

set_alias(*alias: str*) → None

```
# SCPI: SCENario:CPDW:GROup:ALIas
driver.scenario.cpdw.group.set_alias(alias = 'abc')
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

param alias
string

set_select(*select: float*) → None

```
# SCPI: SCENario:CPDW:GROup:SELect
driver.scenario.cpdw.group.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*group: str*) → None

```
# SCPI: SCENario:CPDW:GROup
driver.scenario.cpdw.group.set_value(group = 'abc')
```

Assigns the emitter to one of the available interleaving groups.

param group
string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.cpdw.group.clone()
```

Subgroups

5.26.4.2.1 Add

SCPI Command :

```
SCENario:CPDW:GROup:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CPDW:GROup:ADD
driver.scenario.cpdw.group.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CPDW:GROup:ADD
driver.scenario.cpdw.group.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.5 Csequence

SCPI Commands :

```
SCENario:CSEQuence:ALIas
SCENario:CSEQuence:CLEar
SCENario:CSEQuence:CURRent
SCENario:CSEQuence:DELete
SCENario:CSEQuence:SELect
SCENario:CSEQuence:VARiable
SCENario:CSEQuence
```

class CsequenceCls

Csequence commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:CSEQuence:CLEar
driver.scenario.csequence.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CSEQuence:CLEar
driver.scenario.csequence.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:CSEQuence:DELete
driver.scenario.csequence.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:CSEQuence:ALIas
value: str = driver.scenario.csequence.get_alias()
```

Enters an alias name.

return
alias: string

get_current() → float

```
# SCPI: SCENario:CSEQuence:CURRent
value: float = driver.scenario.csequence.get_current()
```

Sets the sequence/emitter that is used by the scenario.

return
current: float Number of the sequence/emitter in the list with multiple sequences

get_select() → float

```
# SCPI: SCENario:CSEQuence:SELect
value: float = driver.scenario.csequence.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → str

```
# SCPI: SCENario:CSEQuence
value: str = driver.scenario.csequence.get_value()
```

Select an existing sequence, see method RsPulseSeq.Sequence.catalog.

return
csequence: string

get_variable() → str

```
# SCPI: SCENario:CSEQuence:VARiable
value: str = driver.scenario.csequence.get_variable()
```

Sets the collection variable.

return
variable: string

set_alias(*alias: str*) → None

```
# SCPI: SCENario:CSEQuence:ALIas
driver.scenario.csequence.set_alias(alias = 'abc')
```

Enters an alias name.

param alias
string

set_current(*current: float*) → None

```
# SCPI: SCENario:CSEQuence:CURRent
driver.scenario.csequence.set_current(current = 1.0)
```

Sets the sequence/emitter that is used by the scenario.

param current
float Number of the sequence/emitter in the list with multiple sequences

set_select(*select: float*) → None

```
# SCPI: SCENario:CSEQuence:SELect
driver.scenario.csequence.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*csequence: str*) → None

```
# SCPI: SCENario:CSEQuence
driver.scenario.csequence.set_value(csequence = 'abc')
```

Select an existing sequence, see method RsPulseSeq.Sequence.catalog.

param csequence
string

set_variable(*variable: str*) → None

```
# SCPI: SCENario:CSEQuence:VARiable
driver.scenario.csequence.set_variable(variable = 'abc')
```

Sets the collection variable.

param variable
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.csequence.clone()
```

Subgroups

5.26.5.1 Add

SCPI Command :

```
SCENario:CSEQuence:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:CSEQuence:ADD
driver.scenario.csequence.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:CSEQuence:ADD
driver.scenario.csequence.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.6 Destination

SCPI Commands :

```
SCENario:DESTination:CLEar
SCENario:DESTination
```

class DestinationCls

Destination commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:DESTination:CLEar
driver.scenario.destination.clear()
```

No command help available

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DESTination:CLEar
driver.scenario.destination.clear_with_opc()
```

No command help available

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: SCENario:DESTination
value: str = driver.scenario.destination.get_value()
```

Sets the destination for the signal.

return
destination: string Use the command method RsPulseSeq.Destination.Plugin.Variable.catalog to query a list of available export plugins.

set_value(*destination: str*) → None

```
# SCPI: SCENario:DESTination
driver.scenario.destination.set_value(destination = 'abc')
```

Sets the destination for the signal.

param destination
string Use the command method RsPulseSeq.Destination.Plugin.Variable.catalog to query a list of available export plugins.

5.26.7 Df

SCPI Commands :

```
SCENario:DF:ALIas
SCENario:DF:CLEar
SCENario:DF:CURRent
SCENario:DF:DELete
SCENario:DF:DISTance
SCENario:DF:ENABle
SCENario:DF:FREQuency
SCENario:DF:LDELay
SCENario:DF:LEVel
SCENario:DF:PRIority
SCENario:DF:SELect
SCENario:DF:SEQuence
SCENario:DF:THReshold
SCENario:DF:TYPE
```

class DfCls

Df commands group definition. 144 total commands, 14 Subgroups, 14 group commands

clear() → None

```
# SCPI: SCENario:DF:CLEar
driver.scenario.df.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:CLEar
driver.scenario.df.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:DF:DELete
driver.scenario.df.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:DF:ALIas
value: str = driver.scenario.df.get_alias()
```

Enters an alias name.

return
alias: string

get_current() → float

```
# SCPI: SCENario:DF:CURRent
value: float = driver.scenario.df.get_current()
```

Sets the sequence/emitter that is used by the scenario.

return
current: float Number of the sequence/emitter in the list with multiple sequences

get_distance() → float

```
# SCPI: SCENario:DF:DISTance
value: float = driver.scenario.df.get_distance()
```

Sets the distance to the receiver.

return
distance: float Range: 0 to 1e+09, Unit: m

get_enable() → bool

```
# SCPI: SCENario:DF:ENABle
value: bool = driver.scenario.df.get_enable()
```

If enabled, the PDW list is included in the output file.

return
enable: ON| OFF| 1| 0

get_frequency() → float

```
# SCPI: SCENario:DF:FREQuency
value: float = driver.scenario.df.get_frequency()
```

Sets the frequency for the selected emitter.

return
frequency: No help available

get_ldelay() → float

```
# SCPI: SCENario:DF:LDELay
value: float = driver.scenario.df.get_ldelay()
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

return
ldelay: float Range: -1e+09 to 1e+09

get_level() → float

```
# SCPI: SCENario:DF:LEVel
value: float = driver.scenario.df.get_level()
```

Adds a level offset.

return
level: float Range: -200 to 0

get_priority() → float

```
# SCPI: SCENario:DF:PRIority
value: float = driver.scenario.df.get_priority()
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

return
priority: float Range: 1 to 100

get_select() → float

```
# SCPI: SCENario:DF:SELect
value: float = driver.scenario.df.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_sequence() → str

```
# SCPI: SCENario:DF:SEQuence
value: str = driver.scenario.df.get_sequence()
```

Assigns a sequence to the background signal.

return
sequence: string

get_threshold() → float

```
# SCPI: SCENario:DF:THReshold
value: float = driver.scenario.df.get_threshold()
```

Sets a threshold. Pulses at levels below this threshold are omitted.

return
threshold: float Range: -100 to 0

get_type_py() → DfType

```
# SCPI: SCENario:DF:TYPE
value: enums.DfType = driver.scenario.df.get_type_py()
```

Defines whether an emitter/interferer is configured.

return
type_py: EMITter | | WAVeform

set_alias(*alias: str*) → None

```
# SCPI: SCENario:DF:ALIas
driver.scenario.df.set_alias(alias = 'abc')
```

Enters an alias name.

param alias
string

set_current(*current: float*) → None

```
# SCPI: SCENario:DF:CURRent
driver.scenario.df.set_current(current = 1.0)
```

Sets the sequence/emitter that is used by the scenario.

param current
float Number of the sequence/emitter in the list with multiple sequences

set_distance(*distance: float*) → None

```
# SCPI: SCENario:DF:DISTance
driver.scenario.df.set_distance(distance = 1.0)
```

Sets the distance to the receiver.

param distance
float Range: 0 to 1e+09, Unit: m

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:DF:ENABle
driver.scenario.df.set_enable(enable = False)
```

If enabled, the PDW list is included in the output file.

param enable
ON| OFF| 1| 0

set_ldelay(*ldelay: float*) → None

```
# SCPI: SCENario:DF:LDELay
driver.scenario.df.set_ldelay(ldelay = 1.0)
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

param ldelay
float Range: -1e+09 to 1e+09

set_level(*level: float*) → None

```
# SCPI: SCENario:DF:LEVel
driver.scenario.df.set_level(level = 1.0)
```

Adds a level offset.

param level
float Range: -200 to 0

set_priority(*priority: float*) → None

```
# SCPI: SCENario:DF:PRIority
driver.scenario.df.set_priority(priority = 1.0)
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

param priority
float Range: 1 to 100

set_select(*select: float*) → None

```
# SCPI: SCENario:DF:SELect
driver.scenario.df.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_sequence(*sequence: str*) → None

```
# SCPI: SCENario:DF:SEQuence
driver.scenario.df.set_sequence(sequence = 'abc')
```

Assigns a sequence to the background signal.

param sequence
string

set_threshold(*threshold: float*) → None

```
# SCPI: SCENario:DF:THReshold
driver.scenario.df.set_threshold(threshold = 1.0)
```

Sets a threshold. Pulses at levels below this threshold are omitted.

param threshold
float Range: -100 to 0

set_type_py(*type_py: DfType*) → None

```
# SCPI: SCENario:DF:TYPE
driver.scenario.df.set_type_py(type_py = enums.DfType.BACKground)
```

Defines whether an emitter/interferer is configured.

param type_py
EMITter | | WAVeform

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.clone()
```

Subgroups

5.26.7.1 Add

SCPI Command :

```
SCENario:DF:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:DF:ADD
driver.scenario.df.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:ADD
driver.scenario.df.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.2 Direction

SCPI Commands :

```
SCENario:DF:DIRection:PITCh
SCENario:DF:DIRection:ROLL
SCENario:DF:DIRection:TRACk
SCENario:DF:DIRection:YAW
```

class DirectionCls

Direction commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_pitch() → float

```
# SCPI: SCENario:DF:DIRection:PITCh
value: float = driver.scenario.df.direction.get_pitch()
```

Sets the pitch.

return
pitch: float Range: -90 to 90, Unit: grad

get_roll() → float

```
# SCPI: SCENario:DF:DIRection:ROLL
value: float = driver.scenario.df.direction.get_roll()
```

Sets the roll.

return
roll: float Range: 0 to 360

get_track() → bool

```
# SCPI: SCENario:DF:DIRection:TRACk
value: bool = driver.scenario.df.direction.get_track()
```

Turns the antenna in the direction of the receiver.

return
track: ON| OFF| 1| 0

get_yaw() → float

```
# SCPI: SCENario:DF:DIRection:YAW
value: float = driver.scenario.df.direction.get_yaw()
```

Sets the yaw.

return
yaw: float Range: 0 to 360

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:DF:DIRection:PITCh
driver.scenario.df.direction.set_pitch(pitch = 1.0)
```

Sets the pitch.

param pitch
float Range: -90 to 90, Unit: grad

set_roll(*roll: float*) → None

```
# SCPI: SCENario:DF:DIRection:ROLL
driver.scenario.df.direction.set_roll(roll = 1.0)
```

Sets the roll.

param roll
float Range: 0 to 360

set_track(*track: bool*) → None

```
# SCPI: SCENario:DF:DIRection:TRACk
driver.scenario.df.direction.set_track(track = False)
```

Turns the antenna in the direction of the receiver.

param track
ON| OFF| 1| 0

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:DF:DIRection:YAW
driver.scenario.df.direction.set_yaw(yaw = 1.0)
```

Sets the yaw.

param yaw
float Range: 0 to 360

5.26.7.3 Emitter

SCPI Commands :

```
SCENario:DF:EMITter:ENABle
SCENario:DF:EMITter
```

class EmitterCls

Emitter commands group definition. 14 total commands, 2 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SCENario:DF:EMITter:ENABle
value: bool = driver.scenario.df.emitter.get_enable()
```

In a map-based sceanrio, enable selected item for calculation.

return
enable: ON| OFF| 1| 0

get_value() → str

```
# SCPI: SCENario:DF:EMITter
value: str = driver.scenario.df.emitter.get_value()
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

return
emitter: string

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:DF:EMITter:ENABle
driver.scenario.df.emitter.set_enable(enable = False)
```

In a map-based sceanrio, enable selected item for calculation.

param enable
ON| OFF| 1| 0

set_value(*emitter: str*) → None

```
# SCPI: SCENario:DF:EMITter
driver.scenario.df.emitter.set_value(emitter = 'abc')
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

param emitter
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.emitter.clone()
```

Subgroups

5.26.7.3.1 Mode

SCPI Commands :

```
SCENario:DF:EMITter:MODE:BEAM
SCENario:DF:EMITter:MODE:TRACkrec
SCENario:DF:EMITter:MODE
```

class ModeCls

Mode commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_beam() → float

```
# SCPI: SCENario:DF:EMITter:MODE:BEAM
value: float = driver.scenario.df.emitter.mode.get_beam()
```

Sets the used beam of the current mode.

return
beam: float Range: 1 to 32

get_track_rec() → bool

```
# SCPI: SCENario:DF:EMITter:MODE:TRACkrec
value: bool = driver.scenario.df.emitter.mode.get_track_rec()
```

If enabled, the scan follows the receiver automatically.

return
track_rec: ON| OFF| 1| 0

get_value() → float

```
# SCPI: SCENario:DF:EMITter:MODE
value: float = driver.scenario.df.emitter.mode.get_value()
```

Set the emitter mode.

return
mode: float Range: 1 to 32

set_beam(*beam: float*) → None

```
# SCPI: SCENario:DF:EMITter:MODE:BEAM
driver.scenario.df.emitter.mode.set_beam(beam = 1.0)
```

Sets the used beam of the current mode.

param beam
float Range: 1 to 32

set_track_rec(*track_rec: bool*) → None

```
# SCPI: SCENario:DF:EMITter:MODE:TRACkrec
driver.scenario.df.emitter.mode.set_track_rec(track_rec = False)
```

If enabled, the scan follows the receiver automatically.

param track_rec
ON| OFF| 1| 0

set_value(*mode: float*) → None

```
# SCPI: SCENario:DF:EMITter:MODE
driver.scenario.df.emitter.mode.set_value(mode = 1.0)
```

Set the emitter mode.

param mode
float Range: 1 to 32

5.26.7.3.2 State

SCPI Commands :

```
SCENario:DF:EMITter:STATe:CLEar
SCENario:DF:EMITter:STATe:COUNt
SCENario:DF:EMITter:STATe:DELete
SCENario:DF:EMITter:STATe:DURation
SCENario:DF:EMITter:STATe:ENABle
SCENario:DF:EMITter:STATe:INSert
SCENario:DF:EMITter:STATe:SELect
SCENario:DF:EMITter:STATe:VALue
```

class StateCls

State commands group definition. 9 total commands, 1 Subgroups, 8 group commands

clear() → None

```
# SCPI: SCENario:DF:EMITter:STATe:CLEar
driver.scenario.df.emitter.state.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:CLEar
driver.scenario.df.emitter.state.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:DELete
driver.scenario.df.emitter.state.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SCENario:DF:EMITter:STATe:COUNt
value: float = driver.scenario.df.emitter.state.get_count()
```

Queries the number of existing items.

return
count: integer

get_duration() → float

```
# SCPI: SCENario:DF:EMITter:STATe:DURation
value: float = driver.scenario.df.emitter.state.get_duration()
```

Sets the duration during that the emitter remains in the current state.

return
duration: float Range: -1e+06 to 1e+06

get_enable() → bool

```
# SCPI: SCENario:DF:EMITter:STATe:ENABle
value: bool = driver.scenario.df.emitter.state.get_enable()
```

Enables that an emitter can use on and off states.

return
enable: ON| OFF| 1| 0

get_select() → float

```
# SCPI: SCENario:DF:EMITter:STATe:SELect
value: float = driver.scenario.df.emitter.state.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → bool

```
# SCPI: SCENario:DF:EMITter:STATe:VALue
value: bool = driver.scenario.df.emitter.state.get_value()
```

Sets the emitter state during the selected period.

return
value: ON| OFF| 1| 0

set_duration(*duration: float*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:DURation
driver.scenario.df.emitter.state.set_duration(duration = 1.0)
```

Sets the duration during that the emitter remains in the current state.

param duration
float Range: -1e+06 to 1e+06

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:ENABle
driver.scenario.df.emitter.state.set_enable(enable = False)
```

Enables that an emitter can use on and off states.

param enable
ON| OFF| 1| 0

set_insert(*insert: float*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:INSert
driver.scenario.df.emitter.state.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_select(*select: float*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:SELect
driver.scenario.df.emitter.state.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*value: bool*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:VALue
driver.scenario.df.emitter.state.set_value(value = False)
```

Sets the emitter state during the selected period.

param value
ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.emitter.state.clone()
```

Subgroups

5.26.7.3.2.1 Add

SCPI Command :

```
SCENario:DF:EMITter:STATe:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:DF:EMITter:STATe:ADD
driver.scenario.df.emitter.state.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:EMITter:STATe:ADD
driver.scenario.df.emitter.state.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.4 Group

SCPI Commands :

```
SCENario:DF:GROup:ALIas
SCENario:DF:GROup:CATalog
SCENario:DF:GROup:CLEar
SCENario:DF:GROup:COUNt
SCENario:DF:GROup:DELete
SCENario:DF:GROup:SELect
SCENario:DF:GROup
```

class GroupCls

Group commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:DF:GROup:CLEar
driver.scenario.df.group.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:GROup:CLEar
driver.scenario.df.group.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:DF:GROup:DELete
driver.scenario.df.group.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:DF:GROup:ALIas
value: str = driver.scenario.df.group.get_alias()
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

return
alias: string

get_catalog() → str

```
# SCPI: SCENario:DF:GROup:CATalog
value: str = driver.scenario.df.group.get_catalog()
```

Queries the alias names of the configured interleaving groups.

return
catalog: string A list of coma-separated alias names.

get_count() → float

```
# SCPI: SCENario:DF:GROup:COUNt
value: float = driver.scenario.df.group.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:DF:GROup:SELect
value: float = driver.scenario.df.group.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → str

```
# SCPI: SCENario:DF:GROup
value: str = driver.scenario.df.group.get_value()
```

Assigns the emitter to one of the available interleaving groups.

return
group: string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

set_alias(*alias: str*) → None

```
# SCPI: SCENario:DF:GROup:ALIas
driver.scenario.df.group.set_alias(alias = 'abc')
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

param alias
string

set_select(*select: float*) → None

```
# SCPI: SCENario:DF:GROup:SELect
driver.scenario.df.group.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*group: str*) → None

```
# SCPI: SCENario:DF:GROup
driver.scenario.df.group.set_value(group = 'abc')
```

Assigns the emitter to one of the available interleaving groups.

param group
string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.group.clone()
```

Subgroups

5.26.7.4.1 Add

SCPI Command :

```
SCENario:DF:GROup:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:DF:GROup:ADD
driver.scenario.df.group.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:GROup:ADD
driver.scenario.df.group.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.5 Interleaving

SCPI Commands :

```
SCENario:DF:INTerleaving:MODE
SCENario:DF:INTerleaving:FREQagility
SCENario:DF:INTerleaving
```

class InterleavingCls

Interleaving commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_freq_agility() → bool

```
# SCPI: SCENario:DF:INTerleaving:FREQagility
value: bool = driver.scenario.df.interleaving.get_freq_agility()
```

Enables frequency agility in interleaving. Requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

return
freq_agility: ON| OFF| 1| 0

get_mode() → InterleaveMode

```
# SCPI: SCENario:DF:INTerleaving:MODE
value: enums.InterleaveMode = driver.scenario.df.interleaving.get_mode()
```

Select the mode for interleaving.

return
mode: DROP| MERGe DROP Interleaving uses a priority-based dropping algorithm. MERGE Emitters or PDW lists are merged into multiple output files using groups.

get_value() → bool

```
# SCPI: SCENario:DF:INTerleaving
value: bool = driver.scenario.df.interleaving.get_value()
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

return
interleaving: ON| OFF| 1| 0

set_freq_agility(*freq_agility: bool*) → None

```
# SCPI: SCENario:DF:INTerleaving:FREQagility
driver.scenario.df.interleaving.set_freq_agility(freq_agility = False)
```

Enables frequency agility in interleaving. Requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

param freq_agility
ON| OFF| 1| 0

set_mode(*mode: InterleaveMode*) → None

```
# SCPI: SCENario:DF:INTerleaving:MODE
driver.scenario.df.interleaving.set_mode(mode = enums.InterleaveMode.DROP)
```

Select the mode for interleaving.

param mode
DROP| MERGe DROP Interleaving uses a priority-based dropping algorithm. MERGE Emitters or PDW lists are merged into multiple output files using groups.

set_value(*interleaving: bool*) → None

```
# SCPI: SCENario:DF:INTerleaving
driver.scenario.df.interleaving.set_value(interleaving = False)
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

param interleaving
ON| OFF| 1| 0

5.26.7.6 Location

SCPI Commands :

```
SCENario:DF:LOCation:ALTitude
SCENario:DF:LOCation:AZIMuth
SCENario:DF:LOCation:EAST
SCENario:DF:LOCation:ELEVation
SCENario:DF:LOCation:HEIGht
SCENario:DF:LOCation:LATitude
SCENario:DF:LOCation:LONGitude
SCENario:DF:LOCation:NORTh
SCENario:DF:LOCation:PMODe
```

class LocationCls

Location commands group definition. 15 total commands, 3 Subgroups, 9 group commands

get_altitude() → float

```
# SCPI: SCENario:DF:LOCation:ALTitude
value: float = driver.scenario.df.location.get_altitude()
```

Use for defining the altitude of a fixed emitter (no movement) on a georeferenced map.

return
altitude: float Range: -1e+09 to 1e+09

get_azimuth() → float

```
# SCPI: SCENario:DF:LOCation:AZIMuth
value: float = driver.scenario.df.location.get_azimuth()
```

Sets the azimuth.

return
azimuth: float Range: 0 to 360

get_east() → float

```
# SCPI: SCENario:DF:LOCation:EAST
value: float = driver.scenario.df.location.get_east()
```

Sets the emitter coordinates.

return
east: No help available

get_elevation() → float

```
# SCPI: SCENario:DF:LOCation:ELEVation
value: float = driver.scenario.df.location.get_elevation()
```

Sets the elevation.

return
elevation: float Range: -90 to 90

get_height() → float

```
# SCPI: SCENario:DF:LOCation:HEIGht
value: float = driver.scenario.df.location.get_height()
```

Sets the height of the antenna.

return
height: float Range: -1e+09 to 1e+09

get_latitude() → float

```
# SCPI: SCENario:DF:LOCation:LATitude
value: float = driver.scenario.df.location.get_latitude()
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

return
latitude: No help available

get_longitude() → float

```
# SCPI: SCENario:DF:LOCation:LONGitude
value: float = driver.scenario.df.location.get_longitude()
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

return
longitude: float Range: -180 to 180

get_north() → float

```
# SCPI: SCENario:DF:LOCation:NORTh
value: float = driver.scenario.df.location.get_north()
```

Sets the emitter coordinates.

return
north: float Range: -1e+09 to 1e+09, Unit: m

get_pmode() → PmodeLocation

```
# SCPI: SCENario:DF:LOCation:PMODe
value: enums.PmodeLocation = driver.scenario.df.location.get_pmode()
```

Sets if the emitter is static or moving.

return
pmode: STATic| STEPs| MOVing

set_altitude(*altitude: float*) → None

```
# SCPI: SCENario:DF:LOCation:ALTitude
driver.scenario.df.location.set_altitude(altitude = 1.0)
```

Use for defining the altitude of a fixed emitter (no movement) on a georeferenced map.

param altitude
float Range: -1e+09 to 1e+09

set_azimuth(*azimuth: float*) → None

```
# SCPI: SCENario:DF:LOCation:AZIMuth
driver.scenario.df.location.set_azimuth(azimuth = 1.0)
```

Sets the azimuth.

param azimuth
float Range: 0 to 360

set_east(*east: float*) → None

```
# SCPI: SCENario:DF:LOCation:EAST
driver.scenario.df.location.set_east(east = 1.0)
```

Sets the emitter coordinates.

param east
float Range: -1e+09 to 1e+09, Unit: m

set_elevation(*elevation: float*) → None

```
# SCPI: SCENario:DF:LOCation:ELEVation
driver.scenario.df.location.set_elevation(elevation = 1.0)
```

Sets the elevation.

param elevation
float Range: -90 to 90

set_height(*height: float*) → None

```
# SCPI: SCENario:DF:LOCation:HEIGht
driver.scenario.df.location.set_height(height = 1.0)
```

Sets the height of the antenna.

param height
float Range: -1e+09 to 1e+09

set_latitude(*latitude: float*) → None

```
# SCPI: SCENario:DF:LOCation:LATitude
driver.scenario.df.location.set_latitude(latitude = 1.0)
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

param latitude
float Range: -180 to 180

set_longitude(*longitude: float*) → None

```
# SCPI: SCENario:DF:LOCation:LONGitude
driver.scenario.df.location.set_longitude(longitude = 1.0)
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

param longitude
float Range: -180 to 180

set_north(*north: float*) → None

```
# SCPI: SCENario:DF:LOCation:NORTh
driver.scenario.df.location.set_north(north = 1.0)
```

Sets the emitter coordinates.

param north
float Range: -1e+09 to 1e+09, Unit: m

set_pmode(*pmode: PmodeLocation*) → None

```
# SCPI: SCENario:DF:LOCation:PMODe
driver.scenario.df.location.set_pmode(pmode = enums.PmodeLocation.MOVing)
```

Sets if the emitter is static or moving.

param pmode
STATic| STEPs| MOVing

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.location.clone()
```

Subgroups

5.26.7.6.1 Pstep

SCPI Commands :

```
SCENario:DF:LOCation:PSTep:COUNt
SCENario:DF:LOCation:PSTep:DELete
SCENario:DF:LOCation:PSTep:SELect
```

class PstepCls

Pstep commands group definition. 4 total commands, 1 Subgroups, 3 group commands

delete(*delete: float*) → None

```
# SCPI: SCENario:DF:LOCation:PSTep:DELete
driver.scenario.df.location.pstep.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SCENario:DF:LOCation:PSTep:COUNt
value: float = driver.scenario.df.location.pstep.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:DF:LOCation:PSTep:SELect
value: float = driver.scenario.df.location.pstep.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_select(*select: float*) → None

```
# SCPI: SCENario:DF:LOCation:PSTep:SELect
driver.scenario.df.location.pstep.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.location.pstep.clone()
```

Subgroups

5.26.7.6.1.1 Add

SCPI Command :

```
SCENario:DF:LOCation:PSTep:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:DF:LOCation:PSTep:ADD
driver.scenario.df.location.pstep.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:LOCation:PSTep:ADD
driver.scenario.df.location.pstep.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.6.2 Rec

SCPI Command :

```
SCENario:DF:LOCation:REC:PMODe
```

class RecCls

Rec commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_pmode() → Pmode

```
# SCPI: SCENario:DF:LOCation:REC:PMODe
value: enums.Pmode = driver.scenario.df.location.rec.get_pmode()
```

Sets if the receiver is static or moving.

return
pmode: STATic| MOVing

set_pmode(*pmode: Pmode*) → None

```
# SCPI: SCENario:DF:LOCation:REC:PMODe
driver.scenario.df.location.rec.set_pmode(pmode = enums.Pmode.MOVing)
```

Sets if the receiver is static or moving.

param pmode
STATic| MOVing

5.26.7.6.3 Waypoint

SCPI Command :

```
SCENario:DF:LOCation:WAYPoint:CLEar
```

class WaypointCls

Waypoint commands group definition. 1 total commands, 0 Subgroups, 1 group commands

clear() → None

```
# SCPI: SCENario:DF:LOCation:WAYPoint:CLEar
driver.scenario.df.location.waypoint.clear()
```

Discards the selected file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:LOCation:WAYPoint:CLEar
driver.scenario.df.location.waypoint.clear_with_opc()
```

Discards the selected file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.7 Maps

SCPI Commands :

```
SCENario:DF:MAPS:ENABle
SCENario:DF:MAPS:LOAD
```

class MapsCls

Maps commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SCENario:DF:MAPS:ENABle
value: bool = driver.scenario.df.maps.get_enable()
```

Enable maps for the selected scenario. This operation cannot be undone.

return
enable: ON| OFF| 1| 0

load(*load: List[str]*) → None

```
# SCPI: SCENario:DF:MAPS:LOAD
driver.scenario.df.maps.load(load = ['abc1', 'abc2', 'abc3'])
```

This command loads a georeferenced map for the selected scenario. Supported formats:
INTRO_CMD_HELP: Examples of special characters:

- .tif
- .tiff

param load
No help available

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:DF:MAPS:ENABle
driver.scenario.df.maps.set_enable(enable = False)
```

Enable maps for the selected scenario. This operation cannot be undone.

param enable
ON| OFF| 1| 0

5.26.7.8 Marker

SCPI Commands :

```
SCENario:DF:MARKer:AUTO
SCENario:DF:MARKer:FALL
SCENario:DF:MARKer:FORCe
SCENario:DF:MARKer:GATE
SCENario:DF:MARKer:POST
```

(continues on next page)

(continued from previous page)

```
SCENario:DF:MARKer:PRE
SCENario:DF:MARKer:RISE
SCENario:DF:MARKer:WIDTh
```

class MarkerCls

Marker commands group definition. 10 total commands, 1 Subgroups, 8 group commands

get_auto() → float

```
# SCPI: SCENario:DF:MARKer:AUTO
value: float = driver.scenario.df.marker.get_auto()
```

Enables the marker for restart.

return
auto: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_fall() → float

```
# SCPI: SCENario:DF:MARKer:FALL
value: float = driver.scenario.df.marker.get_fall()
```

Enables the marker for fall time.

return
fall: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_force() → bool

```
# SCPI: SCENario:DF:MARKer:FORCe
value: bool = driver.scenario.df.marker.get_force()
```

Determines how the marker is handled.

return
force: ON| OFF| 1| 0 ON | 1 Forces the selected marker type for every pulse of the selected emitter OFF | 0 Leaves the marker unchanged, as defined in the pulses and sequences of this emitter.

get_gate() → float

```
# SCPI: SCENario:DF:MARKer:GATE
value: float = driver.scenario.df.marker.get_gate()
```

Enables marker for gate.

return
gate: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_post() → float

```
# SCPI: SCENario:DF:MARKer:POST
value: float = driver.scenario.df.marker.get_post()
```

Enables marker for post time.

return
post: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_pre() → float

```
# SCPI: SCENario:DF:MARKer:PRE
value: float = driver.scenario.df.marker.get_pre()
```

Enables marker for pre time.

return
pre: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_rise() → float

```
# SCPI: SCENario:DF:MARKer:RISE
value: float = driver.scenario.df.marker.get_rise()
```

Enables marker for rise time.

return
rise: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_width() → float

```
# SCPI: SCENario:DF:MARKer:WIDTh
value: float = driver.scenario.df.marker.get_width()
```

Sets marker for the pulse width.

return
width: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_auto(*auto: float*) → None

```
# SCPI: SCENario:DF:MARKer:AUTO
driver.scenario.df.marker.set_auto(auto = 1.0)
```

Enables the marker for restart.

param auto
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_fall(*fall: float*) → None

```
# SCPI: SCENario:DF:MARKer:FALL
driver.scenario.df.marker.set_fall(fall = 1.0)
```

Enables the marker for fall time.

param fall
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_force(*force: bool*) → None

```
# SCPI: SCENario:DF:MARKer:FORCe
driver.scenario.df.marker.set_force(force = False)
```

Determines how the marker is handled.

param force
ON| OFF| 1| 0 ON | 1 Forces the selected marker type for every pulse of the selected

emitter OFF | 0 Leaves the marker unchanged, as defined in the pulses and sequences of this emitter.

set_gate(*gate: float*) → None

```
# SCPI: SCENario:DF:MARKer:GATE
driver.scenario.df.marker.set_gate(gate = 1.0)
```

Enables marker for gate.

param gate
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_post(*post: float*) → None

```
# SCPI: SCENario:DF:MARKer:POST
driver.scenario.df.marker.set_post(post = 1.0)
```

Enables marker for post time.

param post
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_pre(*pre: float*) → None

```
# SCPI: SCENario:DF:MARKer:PRE
driver.scenario.df.marker.set_pre(pre = 1.0)
```

Enables marker for pre time.

param pre
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_rise(*rise: float*) → None

```
# SCPI: SCENario:DF:MARKer:RISE
driver.scenario.df.marker.set_rise(rise = 1.0)
```

Enables marker for rise time.

param rise
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_width(*width: float*) → None

```
# SCPI: SCENario:DF:MARKer:WIDTh
driver.scenario.df.marker.set_width(width = 1.0)
```

Sets marker for the pulse width.

param width
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.marker.clone()
```

Subgroups

5.26.7.8.1 Time

SCPI Commands :

```
SCENario:DF:MARKer:TIME:POST
SCENario:DF:MARKer:TIME:PRE
```

class TimeCls

Time commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_post() → float

```
# SCPI: SCENario:DF:MARKer:TIME:POST
value: float = driver.scenario.df.marker.time.get_post()
```

Specifies post marker time.

return
post: float Range: 0 to 3600

get_pre() → float

```
# SCPI: SCENario:DF:MARKer:TIME:PRE
value: float = driver.scenario.df.marker.time.get_pre()
```

Specifies pre marker time.

return
pre: float Range: 0 to 3600

set_post(*post: float*) → None

```
# SCPI: SCENario:DF:MARKer:TIME:POST
driver.scenario.df.marker.time.set_post(post = 1.0)
```

Specifies post marker time.

param post
float Range: 0 to 3600

set_pre(*pre: float*) → None

```
# SCPI: SCENario:DF:MARKer:TIME:PRE
driver.scenario.df.marker.time.set_pre(pre = 1.0)
```

Specifies pre marker time.

param pre
float Range: 0 to 3600

5.26.7.9 Mchg

SCPI Commands :

```
SCENario:DF:MCHG:CLEar
SCENario:DF:MCHG:COUNt
SCENario:DF:MCHG:DELete
SCENario:DF:MCHG:SELect
SCENario:DF:MCHG:STARt
SCENario:DF:MCHG:STATe
SCENario:DF:MCHG:STOP
```

class MchgCls

Mchg commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:DF:MCHG:CLEar
driver.scenario.df.mchg.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:MCHG:CLEar
driver.scenario.df.mchg.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:DF:MCHG:DELete
driver.scenario.df.mchg.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SCENario:DF:MCHG:COUNt
value: float = driver.scenario.df.mchg.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:DF:MCHG:SELect
value: float = driver.scenario.df.mchg.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_start() → float

```
# SCPI: SCENario:DF:MCHG:STARt
value: float = driver.scenario.df.mchg.get_start()
```

Sets the start and end time per mode entry.

return
start: No help available

get_state() → bool

```
# SCPI: SCENario:DF:MCHG:STATe
value: bool = driver.scenario.df.mchg.get_state()
```

Enables mode changes.

return
state: ON| OFF| 1| 0

get_stop() → float

```
# SCPI: SCENario:DF:MCHG:STOP
value: float = driver.scenario.df.mchg.get_stop()
```

Sets the start and end time per mode entry.

return
stop: float

set_select(*select: float*) → None

```
# SCPI: SCENario:DF:MCHG:SELect
driver.scenario.df.mchg.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_start(*start: float*) → None

```
# SCPI: SCENario:DF:MCHG:STARt
driver.scenario.df.mchg.set_start(start = 1.0)
```

Sets the start and end time per mode entry.

param start
float

set_state(*state: bool*) → None

```
# SCPI: SCENario:DF:MCHG:STATe
driver.scenario.df.mchg.set_state(state = False)
```

Enables mode changes.

param state
ON| OFF| 1| 0

set_stop(*stop: float*) → None

```
# SCPI: SCENario:DF:MCHG:STOP
driver.scenario.df.mchg.set_stop(stop = 1.0)
```

Sets the start and end time per mode entry.

param stop
float

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.mchg.clone()
```

Subgroups

5.26.7.9.1 Add

SCPI Command :

```
SCENario:DF:MCHG:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:DF:MCHG:ADD
driver.scenario.df.mchg.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:MCHG:ADD
driver.scenario.df.mchg.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.10 Movement

SCPI Commands :

```
SCENario:DF:MOVement:ACCeleration
SCENario:DF:MOVement:ALTitude
SCENario:DF:MOVement:ANGLe
SCENario:DF:MOVement:ATTitude
SCENario:DF:MOVement:CLATitude
SCENario:DF:MOVement:CLEar
SCENario:DF:MOVement:CLONgitude
SCENario:DF:MOVement:EAST
SCENario:DF:MOVement:HEIGht
SCENario:DF:MOVement:LATitude
SCENario:DF:MOVement:LONGitude
SCENario:DF:MOVement:NORTh
SCENario:DF:MOVement:PITCh
SCENario:DF:MOVement:RFRame
SCENario:DF:MOVement:RMODe
SCENario:DF:MOVement:ROLL
SCENario:DF:MOVement:SMOothening
SCENario:DF:MOVement:SPEed
SCENario:DF:MOVement:SPINning
SCENario:DF:MOVement:TYPE
SCENario:DF:MOVement:VEHicle
SCENario:DF:MOVement:WAYPoint
SCENario:DF:MOVement:YAW
```

class MovementCls

Movement commands group definition. 26 total commands, 2 Subgroups, 23 group commands

clear() → None

```
# SCPI: SCENario:DF:MOVement:CLEar
driver.scenario.df.movement.clear()
```

Discards the waypoint and vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:MOVement:CLEar
driver.scenario.df.movement.clear_with_opc()
```

Discards the waypoint and vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_acceleration() → float

```
# SCPI: SCENario:DF:MOVement:ACCeleration
value: float = driver.scenario.df.movement.get_acceleration()
```

Sets the acceleration of the moving emitter.

return
acceleration: float Range: -100 to 100

get_altitude() → float

```
# SCPI: SCENario:DF:MOVement:ALTitude
value: float = driver.scenario.df.movement.get_altitude()
```

Use for defining the altitude of a moving emitter (line trajectory) on a georeferenced map. Use to define the altitude of the end-points of the line.

return
altitude: float Range: -1e+09 to 1e+09

get_angle() → float

```
# SCPI: SCENario:DF:MOVement:ANGLe
value: float = driver.scenario.df.movement.get_angle()
```

Sets the arc angle and thus defines the arc length.

return
angle: float Range: -360 to 360

get_attitude() → Attitude

```
# SCPI: SCENario:DF:MOVement:ATTitude
value: enums.Attitude = driver.scenario.df.movement.get_attitude()
```

Defines how the attitude information is defined.

return
attitude: WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

get_clatitude() → float

```
# SCPI: SCENario:DF:MOVement:CLATitude
value: float = driver.scenario.df.movement.get_clatitude()
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

return
clatitude: No help available

get_clongitude() → float

```
# SCPI: SCENario:DF:MOVement:CLONgitude
value: float = driver.scenario.df.movement.get_clongitude()
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

return
clongitude: float Range: -180 to 180

get_east() → float

```
# SCPI: SCENario:DF:MOVement:EAST
value: float = driver.scenario.df.movement.get_east()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
east: No help available

get_height() → float

```
# SCPI: SCENario:DF:MOVement:HEIGht
value: float = driver.scenario.df.movement.get_height()
```

Sets the height of the emitter at the end of the movement.

return
height: float Range: -1e+09 to 1e+09

get_latitude() → float

```
# SCPI: SCENario:DF:MOVement:LATitude
value: float = driver.scenario.df.movement.get_latitude()
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

return
latitude: No help available

get_longitude() → float

```
# SCPI: SCENario:DF:MOVement:LONGitude
value: float = driver.scenario.df.movement.get_longitude()
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

return
longitude: float Range: -180 to 180

get_north() → float

```
# SCPI: SCENario:DF:MOVement:NORTh
value: float = driver.scenario.df.movement.get_north()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
north: float Range: -1e+09 to 1e+09

get_pitch() → float

```
# SCPI: SCENario:DF:MOVement:PITCh
value: float = driver.scenario.df.movement.get_pitch()
```

Sets the angles of rotation in the corresponding direction.

return
pitch: No help available

get_rframe() → MovementRframe

```
# SCPI: SCENario:DF:MOVement:RFRame
value: enums.MovementRframe = driver.scenario.df.movement.get_rframe()
```

Select the reference frame used to define the emitters coordinates.

return
rframe: WGS| PZ

get_rmode() → MovementRmode

```
# SCPI: SCENario:DF:MOVement:RMODe
value: enums.MovementRmode = driver.scenario.df.movement.get_rmode()
```

Defines the behavior of the moving object when the end of the trajectory is reached.

return
rmode: CYCLic| ROUNdtrip| ONEWay

get_roll() → float

```
# SCPI: SCENario:DF:MOVement:ROLL
value: float = driver.scenario.df.movement.get_roll()
```

Sets the angles of rotation in the corresponding direction.

return
roll: float Range: -180 to 180

get_smoothening() → bool

```
# SCPI: SCENario:DF:MOVement:SMOothening
value: bool = driver.scenario.df.movement.get_smoothening()
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

return
smoothening: ON| OFF| 1| 0

get_speed() → float

```
# SCPI: SCENario:DF:MOVement:SPEed
value: float = driver.scenario.df.movement.get_speed()
```

Sets the speed of the moving emitter.

return
speed: float Range: 0 to 5999

get_spinning() → float

```
# SCPI: SCENario:DF:MOVement:SPINning
value: float = driver.scenario.df.movement.get_spinning()
```

No command help available

return
spinning: No help available

get_type_py() → MovementType

```
# SCPI: SCENario:DF:MOVement:TYPE
value: enums.MovementType = driver.scenario.df.movement.get_type_py()
```

Defines the trajectory shape.

return
type_py: LINE| ARC| WAYPoint| TRACe

get_vehicle() → VehicleMovement

```
# SCPI: SCENario:DF:MOVement:VEHicle
value: enums.VehicleMovement = driver.scenario.df.movement.get_vehicle()
```

Assigns the selected icon.

return
vehicle: LVEHicle| SHIP| AIRPlane| STATionary| DEFault| CAR

get_waypoint() → str

```
# SCPI: SCENario:DF:MOVement:WAYPoint
value: str = driver.scenario.df.movement.get_waypoint()
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

return
waypoint: string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

get_yaw() → float

```
# SCPI: SCENario:DF:MOVement:YAW
value: float = driver.scenario.df.movement.get_yaw()
```

Sets the angles of rotation in the corresponding direction.

return
yaw: No help available

set_acceleration(*acceleration: float*) → None

```
# SCPI: SCENario:DF:MOVement:ACCeleration
driver.scenario.df.movement.set_acceleration(acceleration = 1.0)
```

Sets the acceleration of the moving emitter.

param acceleration
float Range: -100 to 100

set_altitude(*altitude: float*) → None

```
# SCPI: SCENario:DF:MOVement:ALTitude
driver.scenario.df.movement.set_altitude(altitude = 1.0)
```

Use for defining the altitude of a moving emitter (line trajectory) on a georeferenced map. Use to define the altitude of the end-points of the line.

param altitude
float Range: -1e+09 to 1e+09

set_angle(*angle: float*) → None

```
# SCPI: SCENario:DF:MOVement:ANGLe
driver.scenario.df.movement.set_angle(angle = 1.0)
```

Sets the arc angle and thus defines the arc length.

param angle
float Range: -360 to 360

set_attitude(*attitude: Attitude*) → None

```
# SCPI: SCENario:DF:MOVement:ATTitude
driver.scenario.df.movement.set_attitude(attitude = enums.Attitude.CONStant)
```

Defines how the attitude information is defined.

param attitude
WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

set_clatitude(*clatitude: float*) → None

```
# SCPI: SCENario:DF:MOVement:CLATitude
driver.scenario.df.movement.set_clatitude(clatitude = 1.0)
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

param clatitude
float Range: -180 to 180

set_clongitude(*clongitude: float*) → None

```
# SCPI: SCENario:DF:MOVement:CLONgitude
driver.scenario.df.movement.set_clongitude(clongitude = 1.0)
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

param clongitude
float Range: -180 to 180

set_east(*east: float*) → None

```
# SCPI: SCENario:DF:MOVement:EAST
driver.scenario.df.movement.set_east(east = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param east
float Range: -1e+09 to 1e+09

set_height(*height: float*) → None

```
# SCPI: SCENario:DF:MOVement:HEIGht
driver.scenario.df.movement.set_height(height = 1.0)
```

Sets the height of the emitter at the end of the movement.

param height
float Range: -1e+09 to 1e+09

set_latitude(*latitude: float*) → None

```
# SCPI: SCENario:DF:MOVement:LATitude
driver.scenario.df.movement.set_latitude(latitude = 1.0)
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

param latitude
float Range: -180 to 180

set_longitude(*longitude: float*) → None

```
# SCPI: SCENario:DF:MOVement:LONGitude
driver.scenario.df.movement.set_longitude(longitude = 1.0)
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

param longitude
float Range: -180 to 180

set_north(*north: float*) → None

```
# SCPI: SCENario:DF:MOVement:NORTh
driver.scenario.df.movement.set_north(north = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param north
float Range: -1e+09 to 1e+09

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:DF:MOVement:PITCh
driver.scenario.df.movement.set_pitch(pitch = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param pitch
float Range: -180 to 180

set_rframe(*rframe: MovementRframe*) → None

```
# SCPI: SCENario:DF:MOVement:RFRame
driver.scenario.df.movement.set_rframe(rframe = enums.MovementRframe.PZ)
```

Select the reference frame used to define the emitters coordinates.

param rframe
WGS| PZ

set_rmode(*rmode: MovementRmode*) → None

```
# SCPI: SCENario:DF:MOVement:RMODe
driver.scenario.df.movement.set_rmode(rmode = enums.MovementRmode.CYCLic)
```

Defines the behavior of the moving object when the end of the trajectory is reached.

param rmode
CYCLic| ROUNdtrip| ONEWay

set_roll(*roll: float*) → None

```
# SCPI: SCENario:DF:MOVement:ROLL
driver.scenario.df.movement.set_roll(roll = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param roll
float Range: -180 to 180

set_smoothening(*smoothening: bool*) → None

```
# SCPI: SCENario:DF:MOVement:SMOothening
driver.scenario.df.movement.set_smoothening(smoothening = False)
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

param smoothening
ON| OFF| 1| 0

set_speed(*speed: float*) → None

```
# SCPI: SCENario:DF:MOVement:SPEed
driver.scenario.df.movement.set_speed(speed = 1.0)
```

Sets the speed of the moving emitter.

param speed
float Range: 0 to 5999

set_spinning(*spinning: float*) → None

```
# SCPI: SCENario:DF:MOVement:SPINning
driver.scenario.df.movement.set_spinning(spinning = 1.0)
```

No command help available

param spinning
No help available

set_type_py(*type_py: MovementType*) → None

```
# SCPI: SCENario:DF:MOVement:TYPE
driver.scenario.df.movement.set_type_py(type_py = enums.MovementType.ARC)
```

Defines the trajectory shape.

param type_py
LINE| ARC| WAYPoint| TRACe

set_vehicle(*vehicle: VehicleMovement*) → None

```
# SCPI: SCENario:DF:MOVement:VEHicle
driver.scenario.df.movement.set_vehicle(vehicle = enums.VehicleMovement.
↪AIRPlane)
```

Assigns the selected icon.

param vehicle
LVEHicle| SHIP| AIRPlane| STATionary| DEFault| CAR

set_waypoint(*waypoint: str*) → None

```
# SCPI: SCENario:DF:MOVement:WAYPoint
driver.scenario.df.movement.set_waypoint(waypoint = 'abc')
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

param waypoint
string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:DF:MOVement:YAW
driver.scenario.df.movement.set_yaw(yaw = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param yaw
float Range: -180 to 180

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.movement.clone()
```

Subgroups

5.26.7.10.1 ImportPy

SCPI Command :

```
SCENario:DF:MOVement:IMPort
```

class ImportPyCls

ImportPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:DF:MOVement:IMPort
driver.scenario.df.movement.importPy.set()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:MOVement:IMPort
driver.scenario.df.movement.importPy.set_with_opc()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.10.2 Vfile

SCPI Commands :

```
SCENario:DF:MOVement:VFILe:CLEar
SCENario:DF:MOVement:VFILe
```

class VfileCls

Vfile commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:DF:MOVement:VFILe:CLEar
driver.scenario.df.movement.vfile.clear()
```

Discards the selected vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:MOVement:VFILe:CLEar
driver.scenario.df.movement.vfile.clear_with_opc()
```

Discards the selected vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: SCENario:DF:MOVement:VFILe
value: str = driver.scenario.df.movement.vfile.get_value()
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

return
vfile: string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

set_value(*vfile: str*) → None

```
# SCPI: SCENario:DF:MOVement:VFILe
driver.scenario.df.movement.vfile.set_value(vfile = 'abc')
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

param vfile
string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

5.26.7.11 Receiver

SCPI Commands :

```
SCENario:DF:RECeiver:HEIGht
SCENario:DF:RECeiver:LATitude
SCENario:DF:RECeiver:LONGitude
SCENario:DF:RECeiver
```

class ReceiverCls

Receiver commands group definition. 30 total commands, 2 Subgroups, 4 group commands

get_height() → float

```
# SCPI: SCENario:DF:RECeiver:HEIGht
value: float = driver.scenario.df.receiver.get_height()
```

Sets the height of the antenna.

return
height: float Range: -1e+09 to 1e+09

get_latitude() → float

```
# SCPI: SCENario:DF:RECeiver:LATitude
value: float = driver.scenario.df.receiver.get_latitude()
```

Sets the latitude/longitude coordinates of the static receiver.

return
latitude: No help available

get_longitude() → float

```
# SCPI: SCENario:DF:RECeiver:LONGitude
value: float = driver.scenario.df.receiver.get_longitude()
```

Sets the latitude/longitude coordinates of the static receiver.

return
longitude: float Range: -180 to 180

get_value() → str

```
# SCPI: SCENario:DF:RECeiver
value: str = driver.scenario.df.receiver.get_value()
```

Selects an existing receiver, see method RsPulseSeq.Receiver.catalog.

return
receiver: string

set_height(*height: float*) → None

```
# SCPI: SCENario:DF:RECeiver:HEIGht
driver.scenario.df.receiver.set_height(height = 1.0)
```

Sets the height of the antenna.

param height
float Range: -1e+09 to 1e+09

set_latitude(*latitude: float*) → None

```
# SCPI: SCENario:DF:RECeiver:LATitude
driver.scenario.df.receiver.set_latitude(latitude = 1.0)
```

Sets the latitude/longitude coordinates of the static receiver.

param latitude
float Range: -180 to 180

set_longitude(*longitude: float*) → None

```
# SCPI: SCENario:DF:RECeiver:LONGitude
driver.scenario.df.receiver.set_longitude(longitude = 1.0)
```

Sets the latitude/longitude coordinates of the static receiver.

param longitude
float Range: -180 to 180

set_value(*receiver: str*) → None

```
# SCPI: SCENario:DF:RECeiver
driver.scenario.df.receiver.set_value(receiver = 'abc')
```

Selects an existing receiver, see method RsPulseSeq.Receiver.catalog.

param receiver
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.receiver.clone()
```

Subgroups

5.26.7.11.1 Direction

SCPI Commands :

```
SCENario:DF:RECeiver:DIRection:PITCh
SCENario:DF:RECeiver:DIRection:ROLL
SCENario:DF:RECeiver:DIRection:YAW
```

class DirectionCls

Direction commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_pitch() → float

```
# SCPI: SCENario:DF:RECeiver:DIRection:PITCh
value: float = driver.scenario.df.receiver.direction.get_pitch()
```

Sets the pitch.

return
pitch: float Range: -90 to 90, Unit: grad

get_roll() → float

```
# SCPI: SCENario:DF:RECeiver:DIRection:ROLL
value: float = driver.scenario.df.receiver.direction.get_roll()
```

Sets the roll.

return
roll: float Range: 0 to 360

get_yaw() → float

```
# SCPI: SCENario:DF:RECeiver:DIRection:YAW
value: float = driver.scenario.df.receiver.direction.get_yaw()
```

Sets the yaw.

return
yaw: float Range: 0 to 360

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:DF:RECeiver:DIRection:PITCh
driver.scenario.df.receiver.direction.set_pitch(pitch = 1.0)
```

Sets the pitch.

param pitch
float Range: -90 to 90, Unit: grad

set_roll(*roll: float*) → None

```
# SCPI: SCENario:DF:RECeiver:DIRection:ROLL
driver.scenario.df.receiver.direction.set_roll(roll = 1.0)
```

Sets the roll.

param roll
float Range: 0 to 360

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:DF:RECeiver:DIRection:YAW
driver.scenario.df.receiver.direction.set_yaw(yaw = 1.0)
```

Sets the yaw.

param yaw
float Range: 0 to 360

5.26.7.11.2 Movement

SCPI Commands :

```
SCENario:DF:RECeiver:MOVement:ACCeleration
SCENario:DF:RECeiver:MOVement:ANGLe
SCENario:DF:RECeiver:MOVement:ATTitude
SCENario:DF:RECeiver:MOVement:CLEar
SCENario:DF:RECeiver:MOVement:EAST
SCENario:DF:RECeiver:MOVement:HEIGht
SCENario:DF:RECeiver:MOVement:NORTh
SCENario:DF:RECeiver:MOVement:PITCh
SCENario:DF:RECeiver:MOVement:RFRame
SCENario:DF:RECeiver:MOVement:RMODe
SCENario:DF:RECeiver:MOVement:ROLL
```

(continues on next page)

(continued from previous page)

```
SCENario:DF:RECeiver:MOVement:SMOothening
SCENario:DF:RECeiver:MOVement:SPEed
SCENario:DF:RECeiver:MOVement:SPINning
SCENario:DF:RECeiver:MOVement:TYPE
SCENario:DF:RECeiver:MOVement:VEHicle
SCENario:DF:RECeiver:MOVement:YAW
```

class MovementCls

Movement commands group definition. 23 total commands, 4 Subgroups, 17 group commands

clear() → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:CLEar
driver.scenario.df.receiver.movement.clear()
```

Discards the waypoint and vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:CLEar
driver.scenario.df.receiver.movement.clear_with_opc()
```

Discards the waypoint and vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_acceleration() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:ACCeleration
value: float = driver.scenario.df.receiver.movement.get_acceleration()
```

Sets the acceleration of the moving emitter.

return
acceleration: float Range: -100 to 100

get_angle() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:ANGLe
value: float = driver.scenario.df.receiver.movement.get_angle()
```

Sets the arc angle and thus defines the arc length.

return
angle: float Range: -360 to 360

get_attitude() → Attitude

```
# SCPI: SCENario:DF:RECeiver:MOVement:ATTitude
value: enums.Attitude = driver.scenario.df.receiver.movement.get_attitude()
```

Defines how the attitude information is defined.

return
attitude: WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

get_east() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:EAST
value: float = driver.scenario.df.receiver.movement.get_east()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
east: No help available

get_height() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:HEIGht
value: float = driver.scenario.df.receiver.movement.get_height()
```

Sets the height of the emitter at the end of the movement.

return
height: float Range: -1e+09 to 1e+09

get_north() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:NORTh
value: float = driver.scenario.df.receiver.movement.get_north()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
north: float Range: -1e+09 to 1e+09

get_pitch() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:PITCh
value: float = driver.scenario.df.receiver.movement.get_pitch()
```

Sets the angles of rotation in the corresponding direction.

return
pitch: No help available

get_rframe() → MovementRframe

```
# SCPI: SCENario:DF:RECeiver:MOVement:RFRame
value: enums.MovementRframe = driver.scenario.df.receiver.movement.get_rframe()
```

Select the reference frame used to define the emitters coordinates.

return
rframe: WGS| PZ

get_rmode() → MovementRmode

```
# SCPI: SCENario:DF:RECeiver:MOVement:RMODe
value: enums.MovementRmode = driver.scenario.df.receiver.movement.get_rmode()
```

Defines the behavior of the moving object when the end of the trajectory is reached.

return
rmode: CYCLic| ROUNdtrip| ONEWay

get_roll() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:ROLL
value: float = driver.scenario.df.receiver.movement.get_roll()
```

Sets the angles of rotation in the corresponding direction.

return
roll: float Range: -180 to 180

get_smoothening() → bool

```
# SCPI: SCENario:DF:RECeiver:MOVement:SMOothening
value: bool = driver.scenario.df.receiver.movement.get_smoothening()
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

return
smoothening: ON| OFF| 1| 0

get_speed() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:SPEed
value: float = driver.scenario.df.receiver.movement.get_speed()
```

Sets the speed of the moving emitter.

return
speed: float Range: 0 to 5999

get_spinning() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:SPINning
value: float = driver.scenario.df.receiver.movement.get_spinning()
```

No command help available

return
spinning: No help available

get_type_py() → MovementType

```
# SCPI: SCENario:DF:RECeiver:MOVement:TYPE
value: enums.MovementType = driver.scenario.df.receiver.movement.get_type_py()
```

Defines the trajectory shape.

return
type_py: LINE| ARC| WAYPoint| TRACe

get_vehicle() → Vehicle

```
# SCPI: SCENario:DF:RECeiver:MOVement:VEHicle
value: enums.Vehicle = driver.scenario.df.receiver.movement.get_vehicle()
```

Assigns the selected icon.

return
vehicle: LVEHicle| SHIP| AIRPlane| STATionary| RECeiver

get_yaw() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:YAW
value: float = driver.scenario.df.receiver.movement.get_yaw()
```

Sets the angles of rotation in the corresponding direction.

return
yaw: No help available

set_acceleration(*acceleration: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:ACCeleration
driver.scenario.df.receiver.movement.set_acceleration(acceleration = 1.0)
```

Sets the acceleration of the moving emitter.

param acceleration
float Range: -100 to 100

set_angle(*angle: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:ANGLe
driver.scenario.df.receiver.movement.set_angle(angle = 1.0)
```

Sets the arc angle and thus defines the arc length.

param angle
float Range: -360 to 360

set_attitude(*attitude: Attitude*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:ATTitude
driver.scenario.df.receiver.movement.set_attitude(attitude = enums.Attitude.
↪CONStant)
```

Defines how the attitude information is defined.

param attitude
WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

set_east(*east: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:EAST
driver.scenario.df.receiver.movement.set_east(east = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param east
float Range: -1e+09 to 1e+09

set_height(*height: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:HEIGht
driver.scenario.df.receiver.movement.set_height(height = 1.0)
```

Sets the height of the emitter at the end of the movement.

param height
float Range: -1e+09 to 1e+09

set_north(*north: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:NORTh
driver.scenario.df.receiver.movement.set_north(north = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param north
float Range: -1e+09 to 1e+09

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:PITCh
driver.scenario.df.receiver.movement.set_pitch(pitch = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param pitch
float Range: -180 to 180

set_rframe(*rframe: MovementRframe*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:RFRame
driver.scenario.df.receiver.movement.set_rframe(rframe = enums.MovementRframe.
↪PZ)
```

Select the reference frame used to define the emitters coordinates.

param rframe
WGS| PZ

set_rmode(*rmode: MovementRmode*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:RMODe
driver.scenario.df.receiver.movement.set_rmode(rmode = enums.MovementRmode.
↪CYCLic)
```

Defines the behavior of the moving object when the end of the trajectory is reached.

param rmode
CYCLic| ROUNdtrip| ONEWay

set_roll(*roll: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:ROLL
driver.scenario.df.receiver.movement.set_roll(roll = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param roll
float Range: -180 to 180

set_smoothening(*smoothening: bool*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:SMOothening
driver.scenario.df.receiver.movement.set_smoothening(smoothening = False)
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

param smoothening
ON| OFF| 1| 0

set_speed(*speed: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:SPEed
driver.scenario.df.receiver.movement.set_speed(speed = 1.0)
```

Sets the speed of the moving emitter.

param speed
float Range: 0 to 5999

set_spinning(*spinning: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:SPINning
driver.scenario.df.receiver.movement.set_spinning(spinning = 1.0)
```

No command help available

param spinning
No help available

set_type_py(*type_py: MovementType*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:TYPE
driver.scenario.df.receiver.movement.set_type_py(type_py = enums.MovementType.
↪ARC)
```

Defines the trajectory shape.

param type_py
LINE| ARC| WAYPoint| TRACe

set_vehicle(*vehicle: Vehicle*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:VEHicle
driver.scenario.df.receiver.movement.set_vehicle(vehicle = enums.Vehicle.
↪AIRPlane)
```

Assigns the selected icon.

param vehicle
LVEHicle| SHIP| AIRPlane| STATionary| RECeiver

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:YAW
driver.scenario.df.receiver.movement.set_yaw(yaw = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param yaw
float Range: -180 to 180

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.df.receiver.movement.clone()
```

Subgroups

5.26.7.11.2.1 ImportPy

SCPI Command :

```
SCENario:DF:RECeiver:MOVement:IMPort
```

class ImportPyCls

ImportPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:IMPort
driver.scenario.df.receiver.movement.importPy.set()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:IMPort
driver.scenario.df.receiver.movement.importPy.set_with_opc()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.7.11.2.2 Pstep

SCPI Command :

```
SCENario:DF:RECeiver:MOVement:PSTep:SELect
```

class PstepCls

Pstep commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → float

```
# SCPI: SCENario:DF:RECeiver:MOVement:PSTep:SELect
value: float = driver.scenario.df.receiver.movement.pstep.get_select()
```

Selects the specified point on a trace trajectory.

return
select: float

set_select(*select: float*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:PSTep:SELect
driver.scenario.df.receiver.movement.pstep.set_select(select = 1.0)
```

Selects the specified point on a trace trajectory.

param select
float

5.26.7.11.2.3 Vfile

SCPI Commands :

```
SCENario:DF:RECeiver:MOVement:VFILe:CLEar
SCENario:DF:RECeiver:MOVement:VFILe
```

class VfileCls

Vfile commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:VFILe:CLEar
driver.scenario.df.receiver.movement.vfile.clear()
```

Discards the selected vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:VFILe:CLEar
driver.scenario.df.receiver.movement.vfile.clear_with_opc()
```

Discards the selected vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

`get_value()` → str

```
# SCPI: SCENario:DF:RECeiver:MOVement:VFILe
value: str = driver.scenario.df.receiver.movement.vfile.get_value()
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

return
vfile: string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

`set_value`(*vfile: str*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:VFILe
driver.scenario.df.receiver.movement.vfile.set_value(vfile = 'abc')
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

param vfile
string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

5.26.7.11.2.4 Waypoint

SCPI Commands :

```
SCENario:DF:RECeiver:MOVement:WAYPoint:CLEar
SCENario:DF:RECeiver:MOVement:WAYPoint
```

`class WaypointCls`

Waypoint commands group definition. 2 total commands, 0 Subgroups, 2 group commands

`clear()` → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:WAYPoint:CLEar
driver.scenario.df.receiver.movement.waypoint.clear()
```

Discards the selected file.

`clear_with_opc`(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:WAYPoint:CLEar
driver.scenario.df.receiver.movement.waypoint.clear_with_opc()
```

Discards the selected file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

`get_value()` → str

```
# SCPI: SCENario:DF:RECeiver:MOVement:WAYPoint
value: str = driver.scenario.df.receiver.movement.waypoint.get_value()
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

return
waypoint: string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

`set_value(waypoint: str)` → None

```
# SCPI: SCENario:DF:RECeiver:MOVement:WAYPoint
driver.scenario.df.receiver.movement.waypoint.set_value(waypoint = 'abc')
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

param waypoint
string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

5.26.7.12 Subitem

SCPI Commands :

```
SCENario:DF:SUBitem:CURRent
SCENario:DF:SUBitem:SELect
```

`class SubitemCls`

Subitem commands group definition. 2 total commands, 0 Subgroups, 2 group commands

`get_current()` → float

```
# SCPI: SCENario:DF:SUBitem:CURRent
value: float = driver.scenario.df.subitem.get_current()
```

No command help available

return
current: float Range: 1 to 4096

`get_select()` → float

```
# SCPI: SCENario:DF:SUBitem:SELect
value: float = driver.scenario.df.subitem.get_select()
```

No command help available

return
select: float Range: 1 to 4096

set_current(*current: float*) → None

```
# SCPI: SCENario:DF:SUBitem:CURRent
driver.scenario.df.subitem.set_current(current = 1.0)
```

No command help available

param current
float Range: 1 to 4096

set_select(*select: float*) → None

```
# SCPI: SCENario:DF:SUBitem:SELect
driver.scenario.df.subitem.set_select(select = 1.0)
```

No command help available

param select
float Range: 1 to 4096

5.26.7.13 Synchronize

SCPI Command :

```
SCENario:DF:SYNChronize:ENABle
```

class SynchronizeCls

Synchronize commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SCENario:DF:SYNChronize:ENABle
value: bool = driver.scenario.df.synchronize.get_enable()
```

Enables synchronized setup.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:DF:SYNChronize:ENABle
driver.scenario.df.synchronize.set_enable(enable = False)
```

Enables synchronized setup.

param enable
ON| OFF| 1| 0

5.26.7.14 Waveform

SCPI Commands :

```
SCENario:DF:WAVeform:ANTenna
SCENario:DF:WAVeform:EIRP
SCENario:DF:WAVeform:FREQuency
SCENario:DF:WAVeform:LEVel
SCENario:DF:WAVeform:SCAN
SCENario:DF:WAVeform
```

class WaveformCls

Waveform commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_antenna() → str

```
# SCPI: SCENario:DF:WAVeform:ANTenna
value: str = driver.scenario.df.waveform.get_antenna()
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

return
antenna: string

get_eirp() → float

```
# SCPI: SCENario:DF:WAVeform:EIRP
value: float = driver.scenario.df.waveform.get_eirp()
```

Sets the of the interferer.

return
eirp: float Range: -200 to 200

get_frequency() → float

```
# SCPI: SCENario:DF:WAVeform:FREQuency
value: float = driver.scenario.df.waveform.get_frequency()
```

Sets the frequency of the emitter.

return
frequency: float Range: 1000 to 1e+11

get_level() → float

```
# SCPI: SCENario:DF:WAVeform:LEVel
value: float = driver.scenario.df.waveform.get_level()
```

Sets the of the interferer.

return
level: No help available

get_scan() → str

```
# SCPI: SCENario:DF:WAVeform:SCAN
value: str = driver.scenario.df.waveform.get_scan()
```

Assigns an existing antenna scan, see method RsPulseSeq.Scan.catalog.

return
scan: string

get_value() → str

```
# SCPI: SCENario:DF:WAVeform
value: str = driver.scenario.df.waveform.get_value()
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

return
waveform: string

set_antenna(*antenna: str*) → None

```
# SCPI: SCENario:DF:WAVeform:ANTenna
driver.scenario.df.waveform.set_antenna(antenna = 'abc')
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

param antenna
string

set_eirp(*eirp: float*) → None

```
# SCPI: SCENario:DF:WAVeform:EIRP
driver.scenario.df.waveform.set_eirp(eirp = 1.0)
```

Sets the of the interferer.

param eirp
float Range: -200 to 200

set_frequency(*frequency: float*) → None

```
# SCPI: SCENario:DF:WAVeform:FREQuency
driver.scenario.df.waveform.set_frequency(frequency = 1.0)
```

Sets the frequency of the emitter.

param frequency
float Range: 1000 to 1e+11

set_level(*level: float*) → None

```
# SCPI: SCENario:DF:WAVeform:LEVel
driver.scenario.df.waveform.set_level(level = 1.0)
```

Sets the of the interferer.

param level
float Range: -200 to 200

set_scan(*scan: str*) → None

```
# SCPI: SCENario:DF:WAVeform:SCAN
driver.scenario.df.waveform.set_scan(scan = 'abc')
```

Assigns an existing antenna scan, see method RsPulseSeq.Scan.catalog.

param scan
string

set_value(*waveform: str*) → None

```
# SCPI: SCENario:DF:WAVeform
driver.scenario.df.waveform.set_value(waveform = 'abc')
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

param waveform
string

5.26.8 Emitter

SCPI Commands :

```
SCENario:EMITter:CLEar
SCENario:EMITter
```

class EmitterCls

Emitter commands group definition. 7 total commands, 2 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:EMITter:CLEar
driver.scenario.emitter.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:EMITter:CLEar
driver.scenario.emitter.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: SCENario:EMITter
value: str = driver.scenario.emitter.get_value()
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

return
emitter: string

set_value(*emitter: str*) → None

```
# SCPI: SCENario:EMITter
driver.scenario.emitter.set_value(emitter = 'abc')
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

param emitter
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.emitter.clone()
```

Subgroups

5.26.8.1 Direction

SCPI Commands :

```
SCENario:EMITter:DIRection:PITCh
SCENario:EMITter:DIRection:ROLL
SCENario:EMITter:DIRection:YAW
```

class DirectionCls

Direction commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_pitch() → float

```
# SCPI: SCENario:EMITter:DIRection:PITCh
value: float = driver.scenario.emitter.direction.get_pitch()
```

Sets the pitch.

return
pitch: float Range: -90 to 90, Unit: grad

get_roll() → float

```
# SCPI: SCENario:EMITter:DIRection:ROLL
value: float = driver.scenario.emitter.direction.get_roll()
```

Sets the roll.

return
roll: float Range: 0 to 360

get_yaw() → float

```
# SCPI: SCENario:EMITter:DIRection:YAW
value: float = driver.scenario.emitter.direction.get_yaw()
```

Sets the yaw.

return
yaw: float Range: 0 to 360

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:EMITter:DIRection:PITCh
driver.scenario.emitter.direction.set_pitch(pitch = 1.0)
```

Sets the pitch.

param pitch
float Range: -90 to 90, Unit: grad

set_roll(*roll: float*) → None

```
# SCPI: SCENario:EMITter:DIRection:ROLL
driver.scenario.emitter.direction.set_roll(roll = 1.0)
```

Sets the roll.

param roll
float Range: 0 to 360

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:EMITter:DIRection:YAW
driver.scenario.emitter.direction.set_yaw(yaw = 1.0)
```

Sets the yaw.

param yaw
float Range: 0 to 360

5.26.8.2 Mode

SCPI Commands :

```
SCENario:EMITter:MODE:BEAM
SCENario:EMITter:MODE
```

class ModeCls

Mode commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_beam() → float

```
# SCPI: SCENario:EMITter:MODE:BEAM
value: float = driver.scenario.emitter.mode.get_beam()
```

Sets the used beam of the current mode.

return
beam: float Range: 1 to 32

get_value() → float

```
# SCPI: SCENario:EMITter:MODE
value: float = driver.scenario.emitter.mode.get_value()
```

Set the emitter mode.

return
mode: float Range: 1 to 32

set_beam(*beam: float*) → None

```
# SCPI: SCENario:EMITter:MODE:BEAM
driver.scenario.emitter.mode.set_beam(beam = 1.0)
```

Sets the used beam of the current mode.

param beam
float Range: 1 to 32

set_value(*mode: float*) → None

```
# SCPI: SCENario:EMITter:MODE
driver.scenario.emitter.mode.set_value(mode = 1.0)
```

Set the emitter mode.

param mode
float Range: 1 to 32

5.26.9 Generator

SCPI Commands :

```
SCENario:GENerator:CLEar
SCENario:GENerator:PATH
SCENario:GENerator
```

class GeneratorCls

Generator commands group definition. 3 total commands, 0 Subgroups, 3 group commands

clear() → None

```
# SCPI: SCENario:GENerator:CLEar
driver.scenario.generator.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:GENerator:CLEar
driver.scenario.generator.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_path() → float

```
# SCPI: SCENario:GENerator:PATH
value: float = driver.scenario.generator.get_path()
```

Selects the signal path that will play the generated signal.

return
path: float Range: 1 to 32

get_value() → str

```
# SCPI: SCENario:GENerator
value: str = driver.scenario.generator.get_value()
```

Sets the signal generator.

return
generator: string Use the command GENerator:CATalog? to query a list of configured generator.

set_path(*path: float*) → None

```
# SCPI: SCENario:GENerator:PATH
driver.scenario.generator.set_path(path = 1.0)
```

Selects the signal path that will play the generated signal.

param path
float Range: 1 to 32

set_value(*generator: str*) → None

```
# SCPI: SCENario:GENerator
driver.scenario.generator.set_value(generator = 'abc')
```

Sets the signal generator.

param generator
string Use the command GENerator:CATalog? to query a list of configured generator.

5.26.10 IlCache

class IlCacheCls

IlCache commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.ilCache.clone()
```

Subgroups

5.26.10.1 Volatile

SCPI Commands :

```
SCENario:ILCache:VOLatile:CLEar
SCENario:ILCache:VOLatile:VALid
```

class VolatileCls

Volatile commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:ILCache:VOLatile:CLEar
driver.scenario.ilCache.volatile.clear()
```

Deletes the files from the volatile/repository memory.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:ILCache:VOLatile:CLEar
driver.scenario.ilCache.volatile.clear_with_opc()
```

Deletes the files from the volatile/repository memory.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_valid() → bool

```
# SCPI: SCENario:ILCache:VOLatile:VALid
value: bool = driver.scenario.ilCache.volatile.get_valid()
```

Queries whether the volatile/repository memory contains a valid signal file.

return
valid: ON| OFF| 1| 0

5.26.11 Interleave

SCPI Command :

```
SCENario:INTerleave
```

class InterleaveCls

Interleave commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:INTerleave
driver.scenario.interleave.set()
```

If method RsPulseSeq.Scenario.Cemit.Interleaving.value|method RsPulseSeq.Scenario.Cpdw.interleaving|method RsPulseSeq.Scenario.Localized.Interleaving.value|method RsPulseSeq.Scenario.Df.Interleaving.value 1, triggers the calculation of a single output file. The output file comprises the individual PDWs or pulses, where overlapping PDWs or pulses within an interleaving group are dropped, based on a defined priority.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:INTerleave
driver.scenario.interleave.set_with_opc()
```

If method RsPulseSeq.Scenario.Cemit.Interleaving.value|method RsPulseSeq.Scenario.Cpdw.interleaving|method RsPulseSeq.Scenario.Localized.Interleaving.value|method RsPulseSeq.Scenario.Df.Interleaving.value 1, triggers the calculation of a single output file. The output file comprises the individual PDWs or pulses, where overlapping PDWs or pulses within an interleaving group are dropped, based on a defined priority.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12 Localized

SCPI Commands :

```
SCENario:LOCalized:ALIas
SCENario:LOCalized:CLEar
SCENario:LOCalized:CURRent
SCENario:LOCalized:DELete
SCENario:LOCalized:DISTance
SCENario:LOCalized:ENABle
SCENario:LOCalized:FREQuency
SCENario:LOCalized:LDELay
SCENario:LOCalized:LEVel
SCENario:LOCalized:PRIority
SCENario:LOCalized:SELect
SCENario:LOCalized:SEQuence
```

(continues on next page)

(continued from previous page)

```
SCENario:LOCalized:THReshold
SCENario:LOCalized:TYPE
```

class LocalizedCls

Localized commands group definition. 146 total commands, 13 Subgroups, 14 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:CLEar
driver.scenario.localized.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:CLEar
driver.scenario.localized.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:LOCalized:DELete
driver.scenario.localized.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:LOCalized:ALIas
value: str = driver.scenario.localized.get_alias()
```

Enters an alias name.

return
alias: string

get_current() → float

```
# SCPI: SCENario:LOCalized:CURRent
value: float = driver.scenario.localized.get_current()
```

Sets the sequence/emitter that is used by the scenario.

return
current: float Number of the sequence/emitter in the list with multiple sequences

get_distance() → float

```
# SCPI: SCENario:LOCalized:DISTance
value: float = driver.scenario.localized.get_distance()
```

Sets the distance to the receiver.

return
distance: float Range: 0 to 1e+09, Unit: m

get_enable() → bool

```
# SCPI: SCENario:LOCalized:ENABle
value: bool = driver.scenario.localized.get_enable()
```

If enabled, the PDW list is included in the output file.

return
enable: ON| OFF| 1| 0

get_frequency() → float

```
# SCPI: SCENario:LOCalized:FREQuency
value: float = driver.scenario.localized.get_frequency()
```

Sets the frequency for the selected emitter.

return
frequency: No help available

get_ldelay() → float

```
# SCPI: SCENario:LOCalized:LDELay
value: float = driver.scenario.localized.get_ldelay()
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

return
ldelay: float Range: -1e+09 to 1e+09

get_level() → float

```
# SCPI: SCENario:LOCalized:LEVel
value: float = driver.scenario.localized.get_level()
```

Adds a level offset.

return
level: float Range: -200 to 0

get_priority() → float

```
# SCPI: SCENario:LOCalized:PRIority
value: float = driver.scenario.localized.get_priority()
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

return
priority: float Range: 1 to 100

get_select() → float

```
# SCPI: SCENario:LOCalized:SELect
value: float = driver.scenario.localized.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_sequence() → str

```
# SCPI: SCENario:LOCalized:SEQuence
value: str = driver.scenario.localized.get_sequence()
```

Assigns a sequence to the background signal.

return
sequence: string

get_threshold() → float

```
# SCPI: SCENario:LOCalized:THReshold
value: float = driver.scenario.localized.get_threshold()
```

Sets a threshold. Pulses at levels below this threshold are omitted.

return
threshold: float Range: -100 to 0

get_type_py() → DfType

```
# SCPI: SCENario:LOCalized:TYPE
value: enums.DfType = driver.scenario.localized.get_type_py()
```

Defines whether an emitter/interferer is configured.

return
type_py: EMITter | | WAVeform

set_alias(*alias: str*) → None

```
# SCPI: SCENario:LOCalized:ALIas
driver.scenario.localized.set_alias(alias = 'abc')
```

Enters an alias name.

param alias
string

set_current(*current: float*) → None

```
# SCPI: SCENario:LOCalized:CURRent
driver.scenario.localized.set_current(current = 1.0)
```

Sets the sequence/emitter that is used by the scenario.

param current
float Number of the sequence/emitter in the list with multiple sequences

set_distance(*distance: float*) → None

```
# SCPI: SCENario:LOCalized:DISTance
driver.scenario.localized.set_distance(distance = 1.0)
```

Sets the distance to the receiver.

param distance
float Range: 0 to 1e+09, Unit: m

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:LOCalized:ENABle
driver.scenario.localized.set_enable(enable = False)
```

If enabled, the PDW list is included in the output file.

param enable
ON| OFF| 1| 0

set_ldelay(*ldelay: float*) → None

```
# SCPI: SCENario:LOCalized:LDELay
driver.scenario.localized.set_ldelay(ldelay = 1.0)
```

If interleaving is enabled, shifts the processing of the selected PDW list in time.

param ldelay
float Range: -1e+09 to 1e+09

set_level(*level: float*) → None

```
# SCPI: SCENario:LOCalized:LEVel
driver.scenario.localized.set_level(level = 1.0)
```

Adds a level offset.

param level
float Range: -200 to 0

set_priority(*priority: float*) → None

```
# SCPI: SCENario:LOCalized:PRIority
driver.scenario.localized.set_priority(priority = 1.0)
```

Sets the priority of the selected PDW list , where the higher the value the higher the priority.

param priority
float Range: 1 to 100

set_select(*select: float*) → None

```
# SCPI: SCENario:LOCalized:SELect
driver.scenario.localized.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_sequence(*sequence: str*) → None

```
# SCPI: SCENario:LOCalized:SEQuence
driver.scenario.localized.set_sequence(sequence = 'abc')
```

Assigns a sequence to the background signal.

param sequence
string

set_threshold(*threshold: float*) → None

```
# SCPI: SCENario:LOCalized:THReshold
driver.scenario.localized.set_threshold(threshold = 1.0)
```

Sets a threshold. Pulses at levels below this threshold are omitted.

param threshold
float Range: -100 to 0

set_type_py(*type_py: DfType*) → None

```
# SCPI: SCENario:LOCalized:TYPE
driver.scenario.localized.set_type_py(type_py = enums.DfType.BACKground)
```

Defines whether an emitter/interferer is configured.

param type_py
EMITter | | WAVeform

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.clone()
```

Subgroups

5.26.12.1 Add

SCPI Command :

```
SCENario:LOCalized:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:LOCalized:ADD
driver.scenario.localized.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:ADD
driver.scenario.localized.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.2 Direction

SCPI Commands :

```
SCENario:LOCalized:DIRection:PITCh
SCENario:LOCalized:DIRection:ROLL
SCENario:LOCalized:DIRection:TRACk
SCENario:LOCalized:DIRection:YAW
```

class DirectionCls

Direction commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_pitch() → float

```
# SCPI: SCENario:LOCalized:DIRection:PITCh
value: float = driver.scenario.localized.direction.get_pitch()
```

Sets the pitch.

return
pitch: float Range: -90 to 90, Unit: grad

get_roll() → float

```
# SCPI: SCENario:LOCalized:DIRection:ROLL
value: float = driver.scenario.localized.direction.get_roll()
```

Sets the roll.

return
roll: float Range: 0 to 360

get_track() → bool

```
# SCPI: SCENario:LOCalized:DIRection:TRACk
value: bool = driver.scenario.localized.direction.get_track()
```

Turns the antenna in the direction of the receiver.

return
track: ON| OFF| 1| 0

get_yaw() → float

```
# SCPI: SCENario:LOCalized:DIRection:YAW
value: float = driver.scenario.localized.direction.get_yaw()
```

Sets the yaw.

return
yaw: float Range: 0 to 360

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:LOCalized:DIRection:PITCh
driver.scenario.localized.direction.set_pitch(pitch = 1.0)
```

Sets the pitch.

param pitch
float Range: -90 to 90, Unit: grad

set_roll(*roll: float*) → None

```
# SCPI: SCENario:LOCalized:DIRection:ROLL
driver.scenario.localized.direction.set_roll(roll = 1.0)
```

Sets the roll.

param roll
float Range: 0 to 360

set_track(*track: bool*) → None

```
# SCPI: SCENario:LOCalized:DIRection:TRACk
driver.scenario.localized.direction.set_track(track = False)
```

Turns the antenna in the direction of the receiver.

param track
ON| OFF| 1| 0

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:LOCalized:DIRection:YAW
driver.scenario.localized.direction.set_yaw(yaw = 1.0)
```

Sets the yaw.

param yaw
float Range: 0 to 360

5.26.12.3 Emitter

SCPI Commands :

```
SCENario:LOCalized:EMITter:ENABle
SCENario:LOCalized:EMITter
```

class EmitterCls

Emitter commands group definition. 14 total commands, 2 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SCENario:LOCalized:EMITter:ENABle
value: bool = driver.scenario.localized.emitter.get_enable()
```

In a map-based sceanrio, enable selected item for calculation.

return
enable: ON| OFF| 1| 0

get_value() → str

```
# SCPI: SCENario:LOCalized:EMITter
value: str = driver.scenario.localized.emitter.get_value()
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

return
emitter: string

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:LOCalized:EMITter:ENABle
driver.scenario.localized.emitter.set_enable(enable = False)
```

In a map-based sceanrio, enable selected item for calculation.

param enable
ON| OFF| 1| 0

set_value(*emitter: str*) → None

```
# SCPI: SCENario:LOCalized:EMITter
driver.scenario.localized.emitter.set_value(emitter = 'abc')
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

param emitter
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.emitter.clone()
```

Subgroups

5.26.12.3.1 Mode

SCPI Commands :

```
SCENario:LOCalized:EMITter:MODE:BEAM
SCENario:LOCalized:EMITter:MODE:TRACkrec
SCENario:LOCalized:EMITter:MODE
```

class ModeCls

Mode commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_beam() → float

```
# SCPI: SCENario:LOCalized:EMITter:MODE:BEAM
value: float = driver.scenario.localized.emitter.mode.get_beam()
```

Sets the used beam of the current mode.

return
beam: float Range: 1 to 32

get_track_rec() → bool

```
# SCPI: SCENario:LOCalized:EMITter:MODE:TRACkrec
value: bool = driver.scenario.localized.emitter.mode.get_track_rec()
```

If enabled, the scan follows the receiver automatically.

return
track_rec: ON| OFF| 1| 0

get_value() → float

```
# SCPI: SCENario:LOCalized:EMITter:MODE
value: float = driver.scenario.localized.emitter.mode.get_value()
```

Set the emitter mode.

return
mode: float Range: 1 to 32

set_beam(*beam: float*) → None

```
# SCPI: SCENario:LOCalized:EMITter:MODE:BEAM
driver.scenario.localized.emitter.mode.set_beam(beam = 1.0)
```

Sets the used beam of the current mode.

param beam
float Range: 1 to 32

set_track_rec(*track_rec: bool*) → None

```
# SCPI: SCENario:LOCalized:EMITter:MODE:TRACkrec
driver.scenario.localized.emitter.mode.set_track_rec(track_rec = False)
```

If enabled, the scan follows the receiver automatically.

param track_rec
ON| OFF| 1| 0

set_value(*mode: float*) → None

```
# SCPI: SCENario:LOCalized:EMITter:MODE
driver.scenario.localized.emitter.mode.set_value(mode = 1.0)
```

Set the emitter mode.

param mode
float Range: 1 to 32

5.26.12.3.2 State

SCPI Commands :

```
SCENario:LOCalized:EMITter:STATe:CLEar
SCENario:LOCalized:EMITter:STATe:COUNt
SCENario:LOCalized:EMITter:STATe:DELete
SCENario:LOCalized:EMITter:STATe:DURation
SCENario:LOCalized:EMITter:STATe:ENABle
SCENario:LOCalized:EMITter:STATe:INSert
SCENario:LOCalized:EMITter:STATe:SELect
SCENario:LOCalized:EMITter:STATe:VALue
```

class StateCls

State commands group definition. 9 total commands, 1 Subgroups, 8 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:CLEar
driver.scenario.localized.emitter.state.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:CLEar
driver.scenario.localized.emitter.state.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:DELete
driver.scenario.localized.emitter.state.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SCENario:LOCalized:EMITter:STATe:COUNt
value: float = driver.scenario.localized.emitter.state.get_count()
```

Queries the number of existing items.

return
count: integer

get_duration() → float

```
# SCPI: SCENario:LOCalized:EMITter:STATe:DURation
value: float = driver.scenario.localized.emitter.state.get_duration()
```

Sets the duration during that the emitter remains in the current state.

return
duration: float Range: -1e+06 to 1e+06

get_enable() → bool

```
# SCPI: SCENario:LOCalized:EMITter:STATe:ENABle
value: bool = driver.scenario.localized.emitter.state.get_enable()
```

Enables that an emitter can use on and off states.

return
enable: ON| OFF| 1| 0

get_select() → float

```
# SCPI: SCENario:LOCalized:EMITter:STATe:SELect
value: float = driver.scenario.localized.emitter.state.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → bool

```
# SCPI: SCENario:LOCalized:EMITter:STATe:VALue
value: bool = driver.scenario.localized.emitter.state.get_value()
```

Sets the emitter state during the selected period.

return
value: ON| OFF| 1| 0

set_duration(*duration: float*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:DURation
driver.scenario.localized.emitter.state.set_duration(duration = 1.0)
```

Sets the duration during that the emitter remains in the current state.

param duration
float Range: -1e+06 to 1e+06

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:ENABle
driver.scenario.localized.emitter.state.set_enable(enable = False)
```

Enables that an emitter can use on and off states.

param enable
ON| OFF| 1| 0

set_insert(*insert: float*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:INSert
driver.scenario.localized.emitter.state.set_insert(insert = 1.0)
```

Inserts a new item before the selected one.

param insert
float

set_select(*select: float*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:SELect
driver.scenario.localized.emitter.state.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*value: bool*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:VALue
driver.scenario.localized.emitter.state.set_value(value = False)
```

Sets the emitter state during the selected period.

param value
ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.emitter.state.clone()
```

Subgroups

5.26.12.3.2.1 Add

SCPI Command :

```
SCENario:LOCalized:EMITter:STATe:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:ADD
driver.scenario.localized.emitter.state.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:EMITter:STATe:ADD
driver.scenario.localized.emitter.state.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.4 Group

SCPI Commands :

```
SCENario:LOCalized:GROup:ALIas
SCENario:LOCalized:GROup:CATalog
SCENario:LOCalized:GROup:CLEar
SCENario:LOCalized:GROup:COUNt
SCENario:LOCalized:GROup:DELete
SCENario:LOCalized:GROup:SELect
SCENario:LOCalized:GROup
```

class GroupCls

Group commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:GROup:CLEar
driver.scenario.localized.group.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:GROup:CLEar
driver.scenario.localized.group.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:LOCalized:GROup:DELete
driver.scenario.localized.group.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_alias() → str

```
# SCPI: SCENario:LOCalized:GROup:ALIas
value: str = driver.scenario.localized.group.get_alias()
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

return
alias: string

get_catalog() → str

```
# SCPI: SCENario:LOCalized:GROup:CATalog
value: str = driver.scenario.localized.group.get_catalog()
```

Queries the alias names of the configured interleaving groups.

return
catalog: string A list of coma-separated alias names.

get_count() → float

```
# SCPI: SCENario:LOCalized:GROup:COUNt
value: float = driver.scenario.localized.group.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:LOCalized:GROup:SELect
value: float = driver.scenario.localized.group.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_value() → str

```
# SCPI: SCENario:LOCalized:GROup
value: str = driver.scenario.localized.group.get_value()
```

Assigns the emitter to one of the available interleaving groups.

return
group: string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

set_alias(*alias: str*) → None

```
# SCPI: SCENario:LOCalized:GROup:ALIas
driver.scenario.localized.group.set_alias(alias = 'abc')
```

Sets an alias name for the selected interleaving group. See also method RsPulseSeq.Assignment.Group.select.

param alias
string

set_select(*select: float*) → None

```
# SCPI: SCENario:LOCalized:GROup:SELect
driver.scenario.localized.group.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_value(*group: str*) → None

```
# SCPI: SCENario:LOCalized:GROup
driver.scenario.localized.group.set_value(group = 'abc')
```

Assigns the emitter to one of the available interleaving groups.

param group
string Query a list of the alias names of the existing interleaving groups with the command method RsPulseSeq.Scenario.Cpdw.Group.catalog.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.group.clone()
```

Subgroups

5.26.12.4.1 Add

SCPI Command :

```
SCENario:LOCalized:GROup:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:LOCalized:GROup:ADD
driver.scenario.localized.group.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:GROup:ADD
driver.scenario.localized.group.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.5 Interleaving

SCPI Commands :

```
SCENario:LOCalized:INTerleaving:MODE
SCENario:LOCalized:INTerleaving:FREQagility
SCENario:LOCalized:INTerleaving
```

class InterleavingCls

Interleaving commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_freq_agility() → bool

```
# SCPI: SCENario:LOCalized:INTerleaving:FREQagility
value: bool = driver.scenario.localized.interleaving.get_freq_agility()
```

Enables frequency agility in interleaving. Requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

return
freq_agility: ON| OFF| 1| 0

get_mode() → InterleaveMode

```
# SCPI: SCENario:LOCalized:INTerleaving:MODE
value: enums.InterleaveMode = driver.scenario.localized.interleaving.get_mode()
```

Select the mode for interleaving.

return
mode: DROP| MERGe DROP Interleaving uses a priority-based dropping algorithm. MERGE Emitters or PDW lists are merged into multiple output files using groups.

get_value() → bool

```
# SCPI: SCENario:LOCalized:INTerleaving
value: bool = driver.scenario.localized.interleaving.get_value()
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

return
interleaving: ON| OFF| 1| 0

set_freq_agility(*freq_agility: bool*) → None

```
# SCPI: SCENario:LOCalized:INTerleaving:FREQagility
driver.scenario.localized.interleaving.set_freq_agility(freq_agility = False)
```

Enables frequency agility in interleaving. Requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

param freq_agility
ON| OFF| 1| 0

set_mode(*mode: InterleaveMode*) → None

```
# SCPI: SCENario:LOCalized:INTerleaving:MODE
driver.scenario.localized.interleaving.set_mode(mode = enums.InterleaveMode.
↪DROP)
```

Select the mode for interleaving.

param mode
DROP| MERGe DROP Interleaving uses a priority-based dropping algorithm. MERGE Emitters or PDW lists are merged into multiple output files using groups.

set_value(*interleaving: bool*) → None

```
# SCPI: SCENario:LOCalized:INTerleaving
driver.scenario.localized.interleaving.set_value(interleaving = False)
```

If enabled, multiple PDW lists are interleaved into a single output file using a priority-based dropping algorithm. Set the priority with the command method RsPulseSeq.Scenario.Cpdw.priority.

param interleaving
ON| OFF| 1| 0

5.26.12.6 Location

SCPI Commands :

```
SCENario:LOCalized:LOCation:ALTitude
SCENario:LOCalized:LOCation:AZIMuth
SCENario:LOCalized:LOCation:EAST
SCENario:LOCalized:LOCation:ELEVation
SCENario:LOCalized:LOCation:HEIGht
SCENario:LOCalized:LOCation:LATitude
SCENario:LOCalized:LOCation:LONGitude
SCENario:LOCalized:LOCation:NORTh
SCENario:LOCalized:LOCation:PMODe
```

class LocationCls

Location commands group definition. 15 total commands, 3 Subgroups, 9 group commands

get_altitude() → float

```
# SCPI: SCENario:LOCalized:LOCation:ALTitude
value: float = driver.scenario.localized.location.get_altitude()
```

Sets the altitude of the antenna.

return
altitude: float Range: -1e+09 to 1e+09

get_azimuth() → float

```
# SCPI: SCENario:LOCalized:LOCation:AZIMuth
value: float = driver.scenario.localized.location.get_azimuth()
```

Sets the azimuth.

return
azimuth: float Range: 0 to 360

get_east() → float

```
# SCPI: SCENario:LOCalized:LOCation:EAST
value: float = driver.scenario.localized.location.get_east()
```

Sets the emitter coordinates.

return
east: No help available

get_elevation() → float

```
# SCPI: SCENario:LOCalized:LOCation:ELEVation
value: float = driver.scenario.localized.location.get_elevation()
```

Sets the elevation.

return
elevation: float Range: -90 to 90

get_height() → float

```
# SCPI: SCENario:LOCalized:LOCation:HEIGht
value: float = driver.scenario.localized.location.get_height()
```

Sets the height of the antenna.

return
height: float Range: -1e+09 to 1e+09

get_latitude() → float

```
# SCPI: SCENario:LOCalized:LOCation:LATitude
value: float = driver.scenario.localized.location.get_latitude()
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

return
latitude: No help available

get_longitude() → float

```
# SCPI: SCENario:LOCalized:LOCation:LONGitude
value: float = driver.scenario.localized.location.get_longitude()
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

return
longitude: float Range: -180 to 180

get_north() → float

```
# SCPI: SCENario:LOCalized:LOCation:NORTh
value: float = driver.scenario.localized.location.get_north()
```

Sets the emitter coordinates.

return
north: float Range: -1e+09 to 1e+09, Unit: m

get_pmode() → PmodeLocation

```
# SCPI: SCENario:LOCalized:LOCation:PMODe
value: enums.PmodeLocation = driver.scenario.localized.location.get_pmode()
```

Sets if the emitter is static or moving.

return
pmode: STATic| STEPs| MOVing

set_altitude(*altitude: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:ALTitude
driver.scenario.localized.location.set_altitude(altitude = 1.0)
```

Sets the altitude of the antenna.

param altitude
float Range: -1e+09 to 1e+09

set_azimuth(*azimuth: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:AZIMuth
driver.scenario.localized.location.set_azimuth(azimuth = 1.0)
```

Sets the azimuth.

param azimuth
float Range: 0 to 360

set_east(*east: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:EAST
driver.scenario.localized.location.set_east(east = 1.0)
```

Sets the emitter coordinates.

param east
float Range: -1e+09 to 1e+09, Unit: m

set_elevation(*elevation: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:ELEVation
driver.scenario.localized.location.set_elevation(elevation = 1.0)
```

Sets the elevation.

param elevation
float Range: -90 to 90

set_height(*height: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:HEIGht
driver.scenario.localized.location.set_height(height = 1.0)
```

Sets the height of the antenna.

param height
float Range: -1e+09 to 1e+09

set_latitude(*latitude: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:LATitude
driver.scenario.localized.location.set_latitude(latitude = 1.0)
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

param latitude
float Range: -180 to 180

set_longitude(*longitude: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:LONGitude
driver.scenario.localized.location.set_longitude(longitude = 1.0)
```

Use for defining the position of a fixed emitter (no movement) on a georeferenced map. Positive values represent DEGEast. Negative values represent DEGWest.

param longitude
float Range: -180 to 180

set_north(*north: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:NORTh
driver.scenario.localized.location.set_north(north = 1.0)
```

Sets the emitter coordinates.

param north
float Range: -1e+09 to 1e+09, Unit: m

set_pmode(*pmode: PmodeLocation*) → None

```
# SCPI: SCENario:LOCalized:LOCation:PMODe
driver.scenario.localized.location.set_pmode(pmode = enums.PmodeLocation.MOVing)
```

Sets if the emitter is static or moving.

param pmode
STATic| STEPs| MOVing

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.location.clone()
```

Subgroups

5.26.12.6.1 Pstep

SCPI Commands :

```
SCENario:LOCalized:LOCation:PSTep:COUNt
SCENario:LOCalized:LOCation:PSTep:DELete
SCENario:LOCalized:LOCation:PSTep:SELect
```

class PstepCls

Pstep commands group definition. 4 total commands, 1 Subgroups, 3 group commands

delete(*delete: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:PSTep:DELete
driver.scenario.localized.location.pstep.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SCENario:LOCalized:LOCation:PSTep:COUNt
value: float = driver.scenario.localized.location.pstep.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:LOCalized:LOCation:PSTep:SELect
value: float = driver.scenario.localized.location.pstep.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_select(*select: float*) → None

```
# SCPI: SCENario:LOCalized:LOCation:PSTep:SELect
driver.scenario.localized.location.pstep.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.location.pstep.clone()
```

Subgroups

5.26.12.6.1.1 Add

SCPI Command :

```
SCENario:LOCalized:LOCation:PSTep:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:LOCalized:LOCation:PSTep:ADD
driver.scenario.localized.location.pstep.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:LOCation:PSTep:ADD
driver.scenario.localized.location.pstep.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.6.2 Rec

SCPI Command :

```
SCENario:LOCalized:LOCation:REC:PMODe
```

class RecCls

Rec commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_pmode() → Pmode

```
# SCPI: SCENario:LOCalized:LOCation:REC:PMODe
value: enums.Pmode = driver.scenario.localized.location.rec.get_pmode()
```

Sets if the receiver is static or moving.

return
pmode: STATic| MOVing

set_pmode(*pmode: Pmode*) → None

```
# SCPI: SCENario:LOCalized:LOCation:REC:PMODe
driver.scenario.localized.location.rec.set_pmode(pmode = enums.Pmode.MOVing)
```

Sets if the receiver is static or moving.

param pmode
STATic| MOVing

5.26.12.6.3 Waypoint

SCPI Command :

```
SCENario:LOCalized:LOCation:WAYPoint:CLEar
```

class WaypointCls

Waypoint commands group definition. 1 total commands, 0 Subgroups, 1 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:LOCation:WAYPoint:CLEar
driver.scenario.localized.location.waypoint.clear()
```

Discards the selected file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:LOCation:WAYPoint:CLEar
driver.scenario.localized.location.waypoint.clear_with_opc()
```

Discards the selected file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.7 Maps

SCPI Commands :

```
SCENario:LOCalized:MAPS:ENABle
SCENario:LOCalized:MAPS:LOAD
```

class MapsCls

Maps commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SCENario:LOCalized:MAPS:ENABle
value: bool = driver.scenario.localized.maps.get_enable()
```

Enable maps for the selected scenario. This operation cannot be undone.

return
enable: ON| OFF| 1| 0

load(*load: List[str]*) → None

```
# SCPI: SCENario:LOCalized:MAPS:LOAD
driver.scenario.localized.maps.load(load = ['abc1', 'abc2', 'abc3'])
```

This command loads a georeferenced map for the selected scenario. Supported formats:
INTRO_CMD_HELP: Examples of special characters:

- .tif
- .tiff

param load
No help available

`set_enable`(*enable: bool*) → None

```
# SCPI: SCENario:LOCalized:MAPS:ENABle
driver.scenario.localized.maps.set_enable(enable = False)
```

Enable maps for the selected scenario. This operation cannot be undone.

param enable
ON| OFF| 1| 0

5.26.12.8 Marker

SCPI Commands :

```
SCENario:LOCalized:MARKer:AUTO
SCENario:LOCalized:MARKer:FALL
SCENario:LOCalized:MARKer:FORCe
SCENario:LOCalized:MARKer:GATE
SCENario:LOCalized:MARKer:POST
SCENario:LOCalized:MARKer:PRE
SCENario:LOCalized:MARKer:RISE
SCENario:LOCalized:MARKer:WIDTh
```

`class MarkerCls`

Marker commands group definition. 10 total commands, 1 Subgroups, 8 group commands

`get_auto`() → float

```
# SCPI: SCENario:LOCalized:MARKer:AUTO
value: float = driver.scenario.localized.marker.get_auto()
```

Enables the marker for restart.

return
auto: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

`get_fall`() → float

```
# SCPI: SCENario:LOCalized:MARKer:FALL
value: float = driver.scenario.localized.marker.get_fall()
```

Enables the marker for fall time.

return
fall: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_force() → bool

```
# SCPI: SCENario:LOCalized:MARKer:FORCe
value: bool = driver.scenario.localized.marker.get_force()
```

Determines how the marker is handled.

return
force: ON| OFF| 1| 0 ON | 1 Forces the selected marker type for every pulse of the selected emitter OFF | 0 Leaves the marker unchanged, as defined in the pulses and sequences of this emitter.

get_gate() → float

```
# SCPI: SCENario:LOCalized:MARKer:GATE
value: float = driver.scenario.localized.marker.get_gate()
```

Enables marker for gate.

return
gate: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_post() → float

```
# SCPI: SCENario:LOCalized:MARKer:POST
value: float = driver.scenario.localized.marker.get_post()
```

Enables marker for post time.

return
post: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_pre() → float

```
# SCPI: SCENario:LOCalized:MARKer:PRE
value: float = driver.scenario.localized.marker.get_pre()
```

Enables marker for pre time.

return
pre: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_rise() → float

```
# SCPI: SCENario:LOCalized:MARKer:RISE
value: float = driver.scenario.localized.marker.get_rise()
```

Enables marker for rise time.

return
rise: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

get_width() → float

```
# SCPI: SCENario:LOCalized:MARKer:WIDTh
value: float = driver.scenario.localized.marker.get_width()
```

Sets marker for the pulse width.

return
width: float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_auto(*auto: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:AUTO
driver.scenario.localized.marker.set_auto(auto = 1.0)
```

Enables the marker for restart.

param auto
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_fall(*fall: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:FALL
driver.scenario.localized.marker.set_fall(fall = 1.0)
```

Enables the marker for fall time.

param fall
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_force(*force: bool*) → None

```
# SCPI: SCENario:LOCalized:MARKer:FORCe
driver.scenario.localized.marker.set_force(force = False)
```

Determines how the marker is handled.

param force
ON| OFF| 1| 0 ON | 1 Forces the selected marker type for every pulse of the selected emitter OFF | 0 Leaves the marker unchanged, as defined in the pulses and sequences of this emitter.

set_gate(*gate: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:GATE
driver.scenario.localized.marker.set_gate(gate = 1.0)
```

Enables marker for gate.

param gate
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_post(*post: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:POST
driver.scenario.localized.marker.set_post(post = 1.0)
```

Enables marker for post time.

param post
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_pre(*pre: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:PRE
driver.scenario.localized.marker.set_pre(pre = 1.0)
```

Enables marker for pre time.

param pre
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

`set_rise`(*rise: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:RISE
driver.scenario.localized.marker.set_rise(rise = 1.0)
```

Enables marker for rise time.

param rise
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

`set_width`(*width: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:WIDTh
driver.scenario.localized.marker.set_width(width = 1.0)
```

Sets marker for the pulse width.

param width
float Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.marker.clone()
```

Subgroups

5.26.12.8.1 Time

SCPI Commands :

```
SCENario:LOCalized:MARKer:TIME:POST
SCENario:LOCalized:MARKer:TIME:PRE
```

`class TimeCls`

Time commands group definition. 2 total commands, 0 Subgroups, 2 group commands

`get_post`() → float

```
# SCPI: SCENario:LOCalized:MARKer:TIME:POST
value: float = driver.scenario.localized.marker.time.get_post()
```

Specifies post marker time.

return
post: float Range: 0 to 3600

get_pre() → float

```
# SCPI: SCENario:LOCalized:MARKer:TIME:PRE
value: float = driver.scenario.localized.marker.time.get_pre()
```

Specifies pre marker time.

return
pre: float Range: 0 to 3600

set_post(*post: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:TIME:POST
driver.scenario.localized.marker.time.set_post(post = 1.0)
```

Specifies post marker time.

param post
float Range: 0 to 3600

set_pre(*pre: float*) → None

```
# SCPI: SCENario:LOCalized:MARKer:TIME:PRE
driver.scenario.localized.marker.time.set_pre(pre = 1.0)
```

Specifies pre marker time.

param pre
float Range: 0 to 3600

5.26.12.9 Mchg

SCPI Commands :

```
SCENario:LOCalized:MCHG:CLEar
SCENario:LOCalized:MCHG:COUNt
SCENario:LOCalized:MCHG:DELete
SCENario:LOCalized:MCHG:SELect
SCENario:LOCalized:MCHG:STARt
SCENario:LOCalized:MCHG:STATe
SCENario:LOCalized:MCHG:STOP
```

class MchgCls

Mchg commands group definition. 8 total commands, 1 Subgroups, 7 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:MCHG:CLEar
driver.scenario.localized.mchg.clear()
```

Removes all defined modes.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:MCHG:CLEar
driver.scenario.localized.mchg.clear_with_opc()
```

Removes all defined modes.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SCENario:LOCalized:MCHG:DELete
driver.scenario.localized.mchg.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SCENario:LOCalized:MCHG:COUNt
value: float = driver.scenario.localized.mchg.get_count()
```

Queries the number of existing items.

return
count: integer

get_select() → float

```
# SCPI: SCENario:LOCalized:MCHG:SELect
value: float = driver.scenario.localized.mchg.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_start() → float

```
# SCPI: SCENario:LOCalized:MCHG:STARt
value: float = driver.scenario.localized.mchg.get_start()
```

Sets the start and end time per mode entry.

return
start: No help available

get_state() → bool

```
# SCPI: SCENario:LOCalized:MCHG:STATe
value: bool = driver.scenario.localized.mchg.get_state()
```

Enables mode changes.

return
state: ON| OFF| 1| 0

get_stop() → float

```
# SCPI: SCENario:LOCalized:MCHG:STOP
value: float = driver.scenario.localized.mchg.get_stop()
```

Sets the start and end time per mode entry.

return
stop: float

set_select(*select: float*) → None

```
# SCPI: SCENario:LOCalized:MCHG:SELect
driver.scenario.localized.mchg.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_start(*start: float*) → None

```
# SCPI: SCENario:LOCalized:MCHG:STARt
driver.scenario.localized.mchg.set_start(start = 1.0)
```

Sets the start and end time per mode entry.

param start
float

set_state(*state: bool*) → None

```
# SCPI: SCENario:LOCalized:MCHG:STATe
driver.scenario.localized.mchg.set_state(state = False)
```

Enables mode changes.

param state
ON| OFF| 1| 0

set_stop(*stop: float*) → None

```
# SCPI: SCENario:LOCalized:MCHG:STOP
driver.scenario.localized.mchg.set_stop(stop = 1.0)
```

Sets the start and end time per mode entry.

param stop
float

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.mchg.clone()
```

Subgroups

5.26.12.9.1 Add

SCPI Command :

```
SCENario:LOCalized:MCHG:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:LOCalized:MCHG:ADD
driver.scenario.localized.mchg.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:MCHG:ADD
driver.scenario.localized.mchg.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.10 Movement

SCPI Commands :

```
SCENario:LOCalized:MOVement:ACCeleration
SCENario:LOCalized:MOVement:ALTitude
SCENario:LOCalized:MOVement:ANGLe
SCENario:LOCalized:MOVement:ATTitude
SCENario:LOCalized:MOVement:CLATitude
SCENario:LOCalized:MOVement:CLEar
SCENario:LOCalized:MOVement:CLONgitude
SCENario:LOCalized:MOVement:EAST
SCENario:LOCalized:MOVement:HEIGht
SCENario:LOCalized:MOVement:LATitude
SCENario:LOCalized:MOVement:LONGitude
SCENario:LOCalized:MOVement:NORTh
```

(continues on next page)

(continued from previous page)

```
SCENario:LOCalized:MOVement:PITCh
SCENario:LOCalized:MOVement:RFRame
SCENario:LOCalized:MOVement:RMODe
SCENario:LOCalized:MOVement:ROLL
SCENario:LOCalized:MOVement:SMOothening
SCENario:LOCalized:MOVement:SPEed
SCENario:LOCalized:MOVement:SPINning
SCENario:LOCalized:MOVement:TYPE
SCENario:LOCalized:MOVement:VEHicle
SCENario:LOCalized:MOVement:WAYPoint
SCENario:LOCalized:MOVement:YAW
```

class MovementCls

Movement commands group definition. 26 total commands, 2 Subgroups, 23 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:MOVement:CLEar
driver.scenario.localized.movement.clear()
```

Discards the waypoint and vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:MOVement:CLEar
driver.scenario.localized.movement.clear_with_opc()
```

Discards the waypoint and vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_acceleration() → float

```
# SCPI: SCENario:LOCalized:MOVement:ACCeleration
value: float = driver.scenario.localized.movement.get_acceleration()
```

Sets the acceleration of the moving emitter.

return
acceleration: float Range: -100 to 100

get_altitude() → float

```
# SCPI: SCENario:LOCalized:MOVement:ALTitude
value: float = driver.scenario.localized.movement.get_altitude()
```

Use for defining the altitude of a moving emitter (line trajectory) on a georeferenced map. Use to define the altitude of the end-points of the line.

return
altitude: float Range: -1e+09 to 1e+09

get_angle() → float

```
# SCPI: SCENario:LOCalized:MOVement:ANGLe
value: float = driver.scenario.localized.movement.get_angle()
```

Sets the arc angle and thus defines the arc length.

return
angle: float Range: -360 to 360

get_attitude() → Attitude

```
# SCPI: SCENario:LOCalized:MOVement:ATTitude
value: enums.Attitude = driver.scenario.localized.movement.get_attitude()
```

Defines how the attitude information is defined.

return
attitude: WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

get_clatitude() → float

```
# SCPI: SCENario:LOCalized:MOVement:CLATitude
value: float = driver.scenario.localized.movement.get_clatitude()
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

return
clatitude: No help available

get_clongitude() → float

```
# SCPI: SCENario:LOCalized:MOVement:CLONgitude
value: float = driver.scenario.localized.movement.get_clongitude()
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

return
clongitude: float Range: -180 to 180

get_east() → float

```
# SCPI: SCENario:LOCalized:MOVement:EAST
value: float = driver.scenario.localized.movement.get_east()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
east: No help available

get_height() → float

```
# SCPI: SCENario:LOCalized:MOVement:HEIGht
value: float = driver.scenario.localized.movement.get_height()
```

Sets the height of the emitter at the end of the movement.

return
height: float Range: -1e+09 to 1e+09

get_latitude() → float

```
# SCPI: SCENario:LOCalized:MOVement:LATitude
value: float = driver.scenario.localized.movement.get_latitude()
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

return
latitude: No help available

get_longitude() → float

```
# SCPI: SCENario:LOCalized:MOVement:LONGitude
value: float = driver.scenario.localized.movement.get_longitude()
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

return
longitude: float Range: -180 to 180

get_north() → float

```
# SCPI: SCENario:LOCalized:MOVement:NORTh
value: float = driver.scenario.localized.movement.get_north()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
north: float Range: -1e+09 to 1e+09

get_pitch() → float

```
# SCPI: SCENario:LOCalized:MOVement:PITCh
value: float = driver.scenario.localized.movement.get_pitch()
```

Sets the angles of rotation in the corresponding direction.

return
pitch: No help available

get_rframe() → MovementRframe

```
# SCPI: SCENario:LOCalized:MOVement:RFRame
value: enums.MovementRframe = driver.scenario.localized.movement.get_rframe()
```

Select the reference frame used to define the emitters coordinates.

return
rframe: WGS| PZ

get_rmode() → MovementRmode

```
# SCPI: SCENario:LOCalized:MOVement:RMODe
value: enums.MovementRmode = driver.scenario.localized.movement.get_rmode()
```

Defines the behavior of the moving object when the end of the trajectory is reached.

return
rmode: CYCLic| ROUNdtrip| ONEWay

get_roll() → float

```
# SCPI: SCENario:LOCalized:MOVement:ROLL
value: float = driver.scenario.localized.movement.get_roll()
```

Sets the angles of rotation in the corresponding direction.

return
roll: float Range: -180 to 180

get_smoothening() → bool

```
# SCPI: SCENario:LOCalized:MOVement:SMOothening
value: bool = driver.scenario.localized.movement.get_smoothening()
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

return
smoothening: ON| OFF| 1| 0

get_speed() → float

```
# SCPI: SCENario:LOCalized:MOVement:SPEed
value: float = driver.scenario.localized.movement.get_speed()
```

Sets the speed of the moving emitter.

return
speed: float Range: 0 to 5999

get_spinning() → float

```
# SCPI: SCENario:LOCalized:MOVement:SPINning
value: float = driver.scenario.localized.movement.get_spinning()
```

No command help available

return
spinning: No help available

get_type_py() → MovementType

```
# SCPI: SCENario:LOCalized:MOVement:TYPE
value: enums.MovementType = driver.scenario.localized.movement.get_type_py()
```

Defines the trajectory shape.

return
type_py: LINE| ARC| WAYPoint| TRACe

get_vehicle() → VehicleMovement

```
# SCPI: SCENario:LOCalized:MOVement:VEHicle
value: enums.VehicleMovement = driver.scenario.localized.movement.get_vehicle()
```

Assigns the selected icon.

return
vehicle: LVEHicle| SHIP| AIRPlane| STATionary| DEFault| CAR

get_waypoint() → str

```
# SCPI: SCENario:LOCalized:MOVement:WAYPoint
value: str = driver.scenario.localized.movement.get_waypoint()
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

return
waypoint: string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

get_yaw() → float

```
# SCPI: SCENario:LOCalized:MOVement:YAW
value: float = driver.scenario.localized.movement.get_yaw()
```

Sets the angles of rotation in the corresponding direction.

return
yaw: No help available

set_acceleration(*acceleration: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:ACCeleration
driver.scenario.localized.movement.set_acceleration(acceleration = 1.0)
```

Sets the acceleration of the moving emitter.

param acceleration
float Range: -100 to 100

set_altitude(*altitude: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:ALTitude
driver.scenario.localized.movement.set_altitude(altitude = 1.0)
```

Use for defining the altitude of a moving emitter (line trajectory) on a georeferenced map. Use to define the altitude of the end-points of the line.

param altitude
float Range: -1e+09 to 1e+09

set_angle(*angle: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:ANGLe
driver.scenario.localized.movement.set_angle(angle = 1.0)
```

Sets the arc angle and thus defines the arc length.

param angle
float Range: -360 to 360

set_attitude(*attitude: Attitude*) → None

```
# SCPI: SCENario:LOCalized:MOVement:ATTitude
driver.scenario.localized.movement.set_attitude(attitude = enums.Attitude.
↪CONStant)
```

Defines how the attitude information is defined.

param attitude
WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

set_clatitude(*clatitude: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:CLATitude
driver.scenario.localized.movement.set_clatitude(clatitude = 1.0)
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

param clatitude
float Range: -180 to 180

set_clongitude(*clongitude: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:CLONgitude
driver.scenario.localized.movement.set_clongitude(clongitude = 1.0)
```

Use for defining the movement of an emitter (arc trajectory) on a georeferenced map. Use to define the center-point of the arc. Positive values represent DEGEast. Negative values represent DEGWest.

param clongitude
float Range: -180 to 180

set_east(*east: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:EAST
driver.scenario.localized.movement.set_east(east = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param east
float Range: -1e+09 to 1e+09

set_height(*height: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:HEIGht
driver.scenario.localized.movement.set_height(height = 1.0)
```

Sets the height of the emitter at the end of the movement.

param height
float Range: -1e+09 to 1e+09

set_latitude(*latitude: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:LATitude
driver.scenario.localized.movement.set_latitude(latitude = 1.0)
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

param latitude
float Range: -180 to 180

set_longitude(*longitude: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:LONGitude
driver.scenario.localized.movement.set_longitude(longitude = 1.0)
```

Use for defining the movement of an emitter (line trajectory) on a georeferenced map. Use to define the end-points of the line. Positive values represent DEGEast. Negative values represent DEGWest.

param longitude
float Range: -180 to 180

set_north(*north: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:NORTh
driver.scenario.localized.movement.set_north(north = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param north
float Range: -1e+09 to 1e+09

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:PITCh
driver.scenario.localized.movement.set_pitch(pitch = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param pitch
float Range: -180 to 180

set_rframe(*rframe: MovementRframe*) → None

```
# SCPI: SCENario:LOCalized:MOVement:RFRame
driver.scenario.localized.movement.set_rframe(rframe = enums.MovementRframe.PZ)
```

Select the reference frame used to define the emitters coordinates.

param rframe
WGS| PZ

set_rmode(*rmode: MovementRmode*) → None

```
# SCPI: SCENario:LOCalized:MOVement:RMODe
driver.scenario.localized.movement.set_rmode(rmode = enums.MovementRmode.CYCLic)
```

Defines the behavior of the moving object when the end of the trajectory is reached.

param rmode
CYCLic| ROUNdtrip| ONEWay

set_roll(*roll: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:ROLL
driver.scenario.localized.movement.set_roll(roll = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param roll
float Range: -180 to 180

set_smoothening(*smoothening: bool*) → None

```
# SCPI: SCENario:LOCalized:MOVement:SMOothening
driver.scenario.localized.movement.set_smoothening(smoothening = False)
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

param smoothening
ON| OFF| 1| 0

set_speed(*speed: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:SPEed
driver.scenario.localized.movement.set_speed(speed = 1.0)
```

Sets the speed of the moving emitter.

param speed
float Range: 0 to 5999

set_spinning(*spinning: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:SPINning
driver.scenario.localized.movement.set_spinning(spinning = 1.0)
```

No command help available

param spinning
No help available

set_type_py(*type_py: MovementType*) → None

```
# SCPI: SCENario:LOCalized:MOVement:TYPE
driver.scenario.localized.movement.set_type_py(type_py = enums.MovementType.ARC)
```

Defines the trajectory shape.

param type_py
LINE| ARC| WAYPoint| TRACe

set_vehicle(*vehicle: VehicleMovement*) → None

```
# SCPI: SCENario:LOCalized:MOVement:VEHicle
driver.scenario.localized.movement.set_vehicle(vehicle = enums.VehicleMovement.
↪AIRPlane)
```

Assigns the selected icon.

param vehicle
LVEHicle| SHIP| AIRPlane| STATionary| DEFault| CAR

set_waypoint(*waypoint: str*) → None

```
# SCPI: SCENario:LOCalized:MOVement:WAYPoint
driver.scenario.localized.movement.set_waypoint(waypoint = 'abc')
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

param waypoint
string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:LOCalized:MOVement:YAW
driver.scenario.localized.movement.set_yaw(yaw = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param yaw
float Range: -180 to 180

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.movement.clone()
```

Subgroups

5.26.12.10.1 ImportPy

SCPI Command :

```
SCENario:LOCalized:MOVement:IMPort
```

class ImportPyCls

ImportPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:LOCalized:MOVement:IMPort
driver.scenario.localized.movement.importPy.set()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:MOVement:IMPort
driver.scenario.localized.movement.importPy.set_with_opc()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.10.2 Vfile

SCPI Commands :

```
SCENario:LOCalized:MOVement:VFILe:CLEar
SCENario:LOCalized:MOVement:VFILe
```

class VfileCls

Vfile commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:MOVement:VFILe:CLEar
driver.scenario.localized.movement.vfile.clear()
```

Discards the selected vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:MOVement:VFILe:CLEar
driver.scenario.localized.movement.vfile.clear_with_opc()
```

Discards the selected vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: SCENario:LOCalized:MOVement:VFILe
value: str = driver.scenario.localized.movement.vfile.get_value()
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

return
vfile: string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

set_value(*vfile: str*) → None

```
# SCPI: SCENario:LOCalized:MOVement:VFILe
driver.scenario.localized.movement.vfile.set_value(vfile = 'abc')
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

param vfile
string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

5.26.12.11 Receiver

SCPI Commands :

```
SCENario:LOCalized:RECeiver:ANTenna
SCENario:LOCalized:RECeiver:BM
SCENario:LOCalized:RECeiver:GAIN
SCENario:LOCalized:RECeiver:HEIGht
SCENario:LOCalized:RECeiver:LATitude
SCENario:LOCalized:RECeiver:LONGitude
SCENario:LOCalized:RECeiver:SCAN
```

class ReceiverCls

Receiver commands group definition. 33 total commands, 2 Subgroups, 7 group commands

get_antenna() → str

```
# SCPI: SCENario:LOCalized:RECeiver:ANTenna
value: str = driver.scenario.localized.receiver.get_antenna()
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

return
antenna: string

get_bm() → str

```
# SCPI: SCENario:LOCalized:RECeiver:BM
value: str = driver.scenario.localized.receiver.get_bm()
```

No command help available

return
bm: No help available

get_gain() → float

```
# SCPI: SCENario:LOCalized:RECeiver:GAIN
value: float = driver.scenario.localized.receiver.get_gain()
```

Sets the antenna .

return
gain: float Range: -120 to 120

get_height() → float

```
# SCPI: SCENario:LOCalized:RECeiver:HEIGht
value: float = driver.scenario.localized.receiver.get_height()
```

Sets the height of the antenna.

return
height: float Range: -1e+09 to 1e+09

get_latitude() → float

```
# SCPI: SCENario:LOCalized:RECeiver:LATitude
value: float = driver.scenario.localized.receiver.get_latitude()
```

Sets the latitude/longitude coordinates of the static receiver.

return
latitude: No help available

get_longitude() → float

```
# SCPI: SCENario:LOCalized:RECeiver:LONGitude
value: float = driver.scenario.localized.receiver.get_longitude()
```

Sets the latitude/longitude coordinates of the static receiver.

return
longitude: float Range: -180 to 180

get_scan() → str

```
# SCPI: SCENario:LOCalized:RECeiver:SCAN
value: str = driver.scenario.localized.receiver.get_scan()
```

Assigns an existing antenna scan, see method RsPulseSeq.Scan.catalog.

return
scan: string

set_antenna(*antenna: str*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:ANTenna
driver.scenario.localized.receiver.set_antenna(antenna = 'abc')
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

param antenna
string

set_bm(*bm: str*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:BM
driver.scenario.localized.receiver.set_bm(bm = 'abc')
```

No command help available

param bm
No help available

set_gain(*gain: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:GAIN
driver.scenario.localized.receiver.set_gain(gain = 1.0)
```

Sets the antenna .

param gain
float Range: -120 to 120

set_height(*height: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:HEIGht
driver.scenario.localized.receiver.set_height(height = 1.0)
```

Sets the height of the antenna.

param height
float Range: -1e+09 to 1e+09

set_latitude(*latitude: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:LATitude
driver.scenario.localized.receiver.set_latitude(latitude = 1.0)
```

Sets the latitude/longitude coordinates of the static receiver.

param latitude
float Range: -180 to 180

set_longitude(*longitude: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:LONGitude
driver.scenario.localized.receiver.set_longitude(longitude = 1.0)
```

Sets the latitude/longitude coordinates of the static receiver.

param longitude
float Range: -180 to 180

set_scan(*scan: str*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:SCAN
driver.scenario.localized.receiver.set_scan(scan = 'abc')
```

Assigns an existing antenna scan, see method RsPulseSeq.Scan.catalog.

param scan
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.receiver.clone()
```

Subgroups

5.26.12.11.1 Direction

SCPI Commands :

```
SCENario:LOCalized:RECeiver:DIRection:PITCh
SCENario:LOCalized:RECeiver:DIRection:ROLL
SCENario:LOCalized:RECeiver:DIRection:YAW
```

class DirectionCls

Direction commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_pitch() → float

```
# SCPI: SCENario:LOCalized:RECeiver:DIRection:PITCh
value: float = driver.scenario.localized.receiver.direction.get_pitch()
```

Sets the pitch.

return
pitch: float Range: -90 to 90, Unit: grad

get_roll() → float

```
# SCPI: SCENario:LOCalized:RECeiver:DIRection:ROLL
value: float = driver.scenario.localized.receiver.direction.get_roll()
```

Sets the roll.

return
roll: float Range: 0 to 360

get_yaw() → float

```
# SCPI: SCENario:LOCalized:RECeiver:DIRection:YAW
value: float = driver.scenario.localized.receiver.direction.get_yaw()
```

Sets the yaw.

return
yaw: float Range: 0 to 360

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:DIRection:PITCh
driver.scenario.localized.receiver.direction.set_pitch(pitch = 1.0)
```

Sets the pitch.

param pitch
float Range: -90 to 90, Unit: grad

set_roll(*roll: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:DIRection:ROLL
driver.scenario.localized.receiver.direction.set_roll(roll = 1.0)
```

Sets the roll.

param roll
float Range: 0 to 360

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:DIRection:YAW
driver.scenario.localized.receiver.direction.set_yaw(yaw = 1.0)
```

Sets the yaw.

param yaw
float Range: 0 to 360

5.26.12.11.2 Movement

SCPI Commands :

```
SCENario:LOCalized:RECeiver:MOVement:ACCeleration
SCENario:LOCalized:RECeiver:MOVement:ANGLe
SCENario:LOCalized:RECeiver:MOVement:ATTitude
SCENario:LOCalized:RECeiver:MOVement:CLEar
SCENario:LOCalized:RECeiver:MOVement:EAST
SCENario:LOCalized:RECeiver:MOVement:HEIGht
SCENario:LOCalized:RECeiver:MOVement:NORTh
SCENario:LOCalized:RECeiver:MOVement:PITCh
SCENario:LOCalized:RECeiver:MOVement:RFRame
SCENario:LOCalized:RECeiver:MOVement:RMODe
SCENario:LOCalized:RECeiver:MOVement:ROLL
SCENario:LOCalized:RECeiver:MOVement:SMOothening
SCENario:LOCalized:RECeiver:MOVement:SPEed
SCENario:LOCalized:RECeiver:MOVement:SPINning
SCENario:LOCalized:RECeiver:MOVement:TYPE
SCENario:LOCalized:RECeiver:MOVement:VEHicle
SCENario:LOCalized:RECeiver:MOVement:YAW
```

class MovementCls

Movement commands group definition. 23 total commands, 4 Subgroups, 17 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:CLEar
driver.scenario.localized.receiver.movement.clear()
```

Discards the waypoint and vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:CLEar
driver.scenario.localized.receiver.movement.clear_with_opc()
```

Discards the waypoint and vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_acceleration() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ACCeleration
value: float = driver.scenario.localized.receiver.movement.get_acceleration()
```

Sets the acceleration of the moving emitter.

return
acceleration: float Range: -100 to 100

get_angle() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ANGLe
value: float = driver.scenario.localized.receiver.movement.get_angle()
```

Sets the arc angle and thus defines the arc length.

return
angle: float Range: -360 to 360

get_attitude() → Attitude

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ATTitude
value: enums.Attitude = driver.scenario.localized.receiver.movement.get_
↪attitude()
```

Defines how the attitude information is defined.

return
attitude: WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

get_east() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:EAST
value: float = driver.scenario.localized.receiver.movement.get_east()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
east: No help available

get_height() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:HEIGht
value: float = driver.scenario.localized.receiver.movement.get_height()
```

Sets the height of the emitter at the end of the movement.

return
height: float Range: -1e+09 to 1e+09

get_north() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:NORTh
value: float = driver.scenario.localized.receiver.movement.get_north()
```

Sets the East/North coordinates of the emitter at the end of the movement.

return
north: float Range: -1e+09 to 1e+09

get_pitch() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:PITCh
value: float = driver.scenario.localized.receiver.movement.get_pitch()
```

Sets the angles of rotation in the corresponding direction.

return
pitch: No help available

get_rframe() → MovementRframe

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:RFRame
value: enums.MovementRframe = driver.scenario.localized.receiver.movement.get_
↪rframe()
```

Select the reference frame used to define the emitters coordinates.

return
rframe: WGS| PZ

get_rmode() → MovementRmode

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:RMODe
value: enums.MovementRmode = driver.scenario.localized.receiver.movement.get_
↪rmode()
```

Defines the behavior of the moving object when the end of the trajectory is reached.

return
rmode: CYCLic| ROUNdtrip| ONEWay

get_roll() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ROLL
value: float = driver.scenario.localized.receiver.movement.get_roll()
```

Sets the angles of rotation in the corresponding direction.

return
roll: float Range: -180 to 180

get_smoothening() → bool

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:SMOothening
value: bool = driver.scenario.localized.receiver.movement.get_smoothening()
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

return
smoothening: ON| OFF| 1| 0

get_speed() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:SPEed
value: float = driver.scenario.localized.receiver.movement.get_speed()
```

Sets the speed of the moving emitter.

return
speed: float Range: 0 to 5999

get_spinning() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:SPINning
value: float = driver.scenario.localized.receiver.movement.get_spinning()
```

No command help available

return
spinning: No help available

get_type_py() → MovementType

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:TYPE
value: enums.MovementType = driver.scenario.localized.receiver.movement.get_
↪type_py()
```

Defines the trajectory shape.

return
type_py: LINE| ARC| WAYPoint| TRACe

get_vehicle() → Vehicle

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:VEHicle
value: enums.Vehicle = driver.scenario.localized.receiver.movement.get_vehicle()
```

Assigns the selected icon.

return
vehicle: LVEHicle| SHIP| AIRPlane| STATionary| RECeiver

get_yaw() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:YAW
value: float = driver.scenario.localized.receiver.movement.get_yaw()
```

Sets the angles of rotation in the corresponding direction.

return
yaw: No help available

set_acceleration(*acceleration: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ACCeleration
driver.scenario.localized.receiver.movement.set_acceleration(acceleration = 1.0)
```

Sets the acceleration of the moving emitter.

param acceleration
float Range: -100 to 100

set_angle(*angle: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ANGLe
driver.scenario.localized.receiver.movement.set_angle(angle = 1.0)
```

Sets the arc angle and thus defines the arc length.

param angle
float Range: -360 to 360

set_attitude(*attitude: Attitude*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ATTitude
driver.scenario.localized.receiver.movement.set_attitude(attitude = enums.
↪Attitude.CONStant)
```

Defines how the attitude information is defined.

param attitude
WAYPoint| MOTion| CONStant WAYPoint The attitude parameters are extracted from the selected waypoint file. MOTion Enables a constant rate of change of the roll. See method RsPulseSeq.Scenario.Localized.Movement.roll Constant The attitude is constant values.

set_east(*east: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:EAST
driver.scenario.localized.receiver.movement.set_east(east = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param east
float Range: -1e+09 to 1e+09

set_height(*height: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:HEIGht
driver.scenario.localized.receiver.movement.set_height(height = 1.0)
```

Sets the height of the emitter at the end of the movement.

param height
float Range: -1e+09 to 1e+09

set_north(*north: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:NORTh
driver.scenario.localized.receiver.movement.set_north(north = 1.0)
```

Sets the East/North coordinates of the emitter at the end of the movement.

param north
float Range: -1e+09 to 1e+09

set_pitch(*pitch: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:PITCh
driver.scenario.localized.receiver.movement.set_pitch(pitch = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param pitch
float Range: -180 to 180

set_rframe(*rframe: MovementRframe*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:RFRame
driver.scenario.localized.receiver.movement.set_rframe(rframe = enums.
↪MovementRframe.PZ)
```

Select the reference frame used to define the emitters coordinates.

param rframe
WGS| PZ

set_rmode(*rmode: MovementRmode*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:RMODe
driver.scenario.localized.receiver.movement.set_rmode(rmode = enums.
↪MovementRmode.CYCLic)
```

Defines the behavior of the moving object when the end of the trajectory is reached.

param rmode
CYCLic| ROUNdtrip| ONEWay

set_roll(*roll: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:ROLL
driver.scenario.localized.receiver.movement.set_roll(roll = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param roll
float Range: -180 to 180

set_smoothening(*smoothening: bool*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:SMOothening
driver.scenario.localized.receiver.movement.set_smoothening(smoothening = False)
```

If a vehicle description file is loaded, activates smoothening. See method RsPulseSeq.Scenario.Localized.Movement.Vfile. value.

param smoothening
ON| OFF| 1| 0

set_speed(*speed: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:SPEed
driver.scenario.localized.receiver.movement.set_speed(speed = 1.0)
```

Sets the speed of the moving emitter.

param speed
float Range: 0 to 5999

set_spinning(*spinning: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:SPINning
driver.scenario.localized.receiver.movement.set_spinning(spinning = 1.0)
```

No command help available

param spinning
No help available

set_type_py(*type_py: MovementType*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:TYPE
driver.scenario.localized.receiver.movement.set_type_py(type_py = enums.
↪MovementType.ARC)
```

Defines the trajectory shape.

param type_py
LINE| ARC| WAYPoint| TRACe

set_vehicle(*vehicle: Vehicle*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:VEHicle
driver.scenario.localized.receiver.movement.set_vehicle(vehicle = enums.Vehicle.
↪AIRPlane)
```

Assigns the selected icon.

param vehicle
LVEHicle| SHIP| AIRPlane| STATionary| RECeiver

set_yaw(*yaw: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:YAW
driver.scenario.localized.receiver.movement.set_yaw(yaw = 1.0)
```

Sets the angles of rotation in the corresponding direction.

param yaw
float Range: -180 to 180

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.localized.receiver.movement.clone()
```

Subgroups

5.26.12.11.2.1 ImportPy

SCPI Command :

```
SCENario:LOCalized:RECeiver:MOVement:IMPort
```

class ImportPyCls

ImportPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:IMPort
driver.scenario.localized.receiver.movement.importPy.set()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:IMPort
driver.scenario.localized.receiver.movement.importPy.set_with_opc()
```

Imports the selected waypoint and vehicle description files into the repository and applies them.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.12.11.2.2 Pstep

SCPI Command :

```
SCENario:LOCalized:RECeiver:MOVement:PSTep:SELect
```

class PstepCls

Pstep commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → float

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:PSTep:SELect
value: float = driver.scenario.localized.receiver.movement.pstep.get_select()
```

Selects the specified point on a trace trajectory.

return
select: float

set_select(*select: float*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:PSTep:SELect
driver.scenario.localized.receiver.movement.pstep.set_select(select = 1.0)
```

Selects the specified point on a trace trajectory.

param select
float

5.26.12.11.2.3 Vfile

SCPI Commands :

```
SCENario:LOCalized:RECeiver:MOVement:VFILe:CLEar
SCENario:LOCalized:RECeiver:MOVement:VFILe
```

class VfileCls

Vfile commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:VFILe:CLEar
driver.scenario.localized.receiver.movement.vfile.clear()
```

Discards the selected vehicle description file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:VFILe:CLEar
driver.scenario.localized.receiver.movement.vfile.clear_with_opc()
```

Discards the selected vehicle description file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:VFILe
value: str = driver.scenario.localized.receiver.movement.vfile.get_value()
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

return
vfile: string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

set_value(*vfile: str*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:VFILe
driver.scenario.localized.receiver.movement.vfile.set_value(vfile = 'abc')
```

Loads the selected vehicle description file (*.xvd) . To import and apply the files, send the command method RsPulseSeq. Scenario.Localized.Movement.ImportPy.set.

param vfile
string Filename or complete file path, incl. file extension. Example files are provided with the software. For description, see 'Vehicle description files (Used for smoothening) '.

5.26.12.11.2.4 Waypoint

SCPI Commands :

```
SCENario:LOCalized:RECeiver:MOVement:WAYPoint:CLEar
SCENario:LOCalized:RECeiver:MOVement:WAYPoint
```

class WaypointCls

Waypoint commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:WAYPoint:CLEar
driver.scenario.localized.receiver.movement.waypoint.clear()
```

Discards the selected file.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:WAYPoint:CLEar
driver.scenario.localized.receiver.movement.waypoint.clear_with_opc()
```

Discards the selected file.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:WAYPoint
value: str = driver.scenario.localized.receiver.movement.waypoint.get_value()
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

return
waypoint: string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

set_value(*waypoint: str*) → None

```
# SCPI: SCENario:LOCalized:RECeiver:MOVement:WAYPoint
driver.scenario.localized.receiver.movement.waypoint.set_value(waypoint = 'abc')
```

Loads the selected waypoint file. To import and apply the files, send the command method RsPulseSeq.Scenario.Localized. Movement.ImportPy.set.

param waypoint
string Filename or complete file path, incl. file extension. Waypoint files must have the extension *.txt, *.kml or *.xtd. Example files are provided with the software. For description, see 'Movement files'.

5.26.12.12 Subitem

SCPI Commands :

```
SCENario:LOCalized:SUBitem:CURRent
SCENario:LOCalized:SUBitem:SELect
```

class SubitemCls

Subitem commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_current() → float

```
# SCPI: SCENario:LOCalized:SUBitem:CURRent
value: float = driver.scenario.localized.subitem.get_current()
```

No command help available

return
current: float Range: 1 to 4096

get_select() → float

```
# SCPI: SCENario:LOCalized:SUBitem:SELect
value: float = driver.scenario.localized.subitem.get_select()
```

No command help available

return
select: float Range: 1 to 4096

set_current(*current: float*) → None

```
# SCPI: SCENario:LOCalized:SUBitem:CURRent
driver.scenario.localized.subitem.set_current(current = 1.0)
```

No command help available

param current
float Range: 1 to 4096

set_select(*select: float*) → None

```
# SCPI: SCENario:LOCalized:SUBitem:SELect
driver.scenario.localized.subitem.set_select(select = 1.0)
```

No command help available

param select
float Range: 1 to 4096

5.26.12.13 Waveform

SCPI Commands :

```
SCENario:LOCalized:WAVeform:ANTenna
SCENario:LOCalized:WAVeform:EIRP
SCENario:LOCalized:WAVeform:FREQuency
SCENario:LOCalized:WAVeform:LEVel
SCENario:LOCalized:WAVeform:SCAN
SCENario:LOCalized:WAVeform
```

class WaveformCls

Waveform commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_antenna() → str

```
# SCPI: SCENario:LOCalized:WAVeform:ANTenna
value: str = driver.scenario.localized.waveform.get_antenna()
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

return
antenna: string

get_eirp() → float

```
# SCPI: SCENario:LOCalized:WAVeform:EIRP
value: float = driver.scenario.localized.waveform.get_eirp()
```

Sets the of the interferer.

return
eirp: float Range: -200 to 200

get_frequency() → float

```
# SCPI: SCENario:LOCalized:WAVeform:FREQuency
value: float = driver.scenario.localized.waveform.get_frequency()
```

Sets the frequency of the emitter.

return
frequency: float Range: 1000 to 1e+11

get_level() → float

```
# SCPI: SCENario:LOCalized:WAVeform:LEVel
value: float = driver.scenario.localized.waveform.get_level()
```

Sets the of the interferer.

return
level: No help available

get_scan() → str

```
# SCPI: SCENario:LOCalized:WAVeform:SCAN
value: str = driver.scenario.localized.waveform.get_scan()
```

Assigns an existing antenna scan, see method RsPulseSeq.Scan.catalog.

return
scan: string

get_value() → str

```
# SCPI: SCENario:LOCalized:WAVeform
value: str = driver.scenario.localized.waveform.get_value()
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

return
waveform: string

set_antenna(*antenna: str*) → None

```
# SCPI: SCENario:LOCalized:WAVeform:ANTenna
driver.scenario.localized.waveform.set_antenna(antenna = 'abc')
```

Assigns an existing antenna pattern, see method RsPulseSeq.Antenna.catalog.

param antenna
string

set_eirp(*eirp: float*) → None

```
# SCPI: SCENario:LOCalized:WAVeform:EIRP
driver.scenario.localized.waveform.set_eirp(eirp = 1.0)
```

Sets the of the interferer.

param eirp
float Range: -200 to 200

set_frequency(*frequency: float*) → None

```
# SCPI: SCENario:LOCalized:WAVeform:FREQuency
driver.scenario.localized.waveform.set_frequency(frequency = 1.0)
```

Sets the frequency of the emitter.

param frequency
float Range: 1000 to 1e+11

set_level(*level: float*) → None

```
# SCPI: SCENario:LOCalized:WAVeform:LEVel
driver.scenario.localized.waveform.set_level(level = 1.0)
```

Sets the of the interferer.

param level
float Range: -200 to 200

set_scan(*scan: str*) → None

```
# SCPI: SCENario:LOCalized:WAVeform:SCAN
driver.scenario.localized.waveform.set_scan(scan = 'abc')
```

Assigns an existing antenna scan, see method RsPulseSeq.Scan.catalog.

param scan
string

set_value(*waveform: str*) → None

```
# SCPI: SCENario:LOCalized:WAVeform
driver.scenario.localized.waveform.set_value(waveform = 'abc')
```

Assigns an existing emitter or an existing waveform, see method RsPulseSeq.Waveform.catalog and method RsPulseSeq.Emitter. catalog.

param waveform
string

5.26.13 Output

SCPI Commands :

```
SCENario:OUTPut:CLIPping
SCENario:OUTPut:FORMat
SCENario:OUTPut:FREQuency
SCENario:OUTPut:LEVel
SCENario:OUTPut:MTMode
SCENario:OUTPut:MTTHreads
SCENario:OUTPut:MULTithread
SCENario:OUTPut:PATH
SCENario:OUTPut:RUNMode
SCENario:OUTPut:TARGet
SCENario:OUTPut:THReshold
```

class OutputCls

Output commands group definition. 27 total commands, 7 Subgroups, 11 group commands

get_clipping() → float

```
# SCPI: SCENario:OUTPut:CLIPping
value: float = driver.scenario.output.get_clipping()
```

Sets a maximum level to limit the dynamic range of the signal. Pulses at levels above this threshold are reduced (clipped) to the configured level.

return
clipping: float Range: -100 to 20, Unit: dBm

get_format_py() → OutFormat

```
# SCPI: SCENario:OUTPut:FORMat
value: enums.OutFormat = driver.scenario.output.get_format_py()
```

Sets the type of the generated waveform file.

return
format_py: WV| MSW

get_frequency() → float

```
# SCPI: SCENario:OUTPut:FREQuency
value: float = driver.scenario.output.get_frequency()
```

Sets the carrier RF frequency of the generated signal.

return
frequency: float Range: 1000 to 1e+11

get_level() → float

```
# SCPI: SCENario:OUTPut:LEVel
value: float = driver.scenario.output.get_level()
```

Sets the reference level used by the calculation of the pulse envelope.

return
level: float Range: -130 to 30

get_mt_mode() → AutoManualMode

```
# SCPI: SCENario:OUTPut:MTMode
value: enums.AutoManualMode = driver.scenario.output.get_mt_mode()
```

If multithreading is enabled with method RsPulseSeq.Scenario.Output.multithread, sets the mode to use for multithreading.

return
mt_mode: AUTO| MANual

get_mt_threads() → float

```
# SCPI: SCENario:OUTPut:MTTHreads
value: float = driver.scenario.output.get_mt_threads()
```

In manual mode, sets the required number of threads for the signal calculation.

return
mt_threads: float Range: 0 to 1000

get_multithread() → bool

```
# SCPI: SCENario:OUTPut:MULTithread
value: bool = driver.scenario.output.get_multithread()
```

Enable to optmize the calculation speed.

return
multithread: ON| OFF| 1| 0

get_path() → str

```
# SCPI: SCENario:OUTPut:PATH
value: str = driver.scenario.output.get_path()
```

Sets the directory the generated waveform is stored in.

return
path: string File path

get_run_mode() → RepeatMode

```
# SCPI: SCENario:OUTPut:RUNMode
value: enums.RepeatMode = driver.scenario.output.get_run_mode()
```

Defines the way the generated signal is processed.

return
run_mode: CONTinuous| SINGle

get_target() → TargetOut

```
# SCPI: SCENario:OUTPut:TARGet
value: enums.TargetOut = driver.scenario.output.get_target()
```

Defines whether the software creates an ARB file or transfers the generated waveform to a connected physical generator. To assign a generator, use the command method RsPulseSeq.Scenario.Generator.value. To set the name and the directory the ARB file is stored in, use the command method RsPulseSeq.Scenario.Output.path.

return
target: INSTrument| FILE

get_threshold() → float

```
# SCPI: SCENario:OUTPut:THReshold
value: float = driver.scenario.output.get_threshold()
```

Sets a threshold. Pulses at levels below this threshold are omitted.

return
threshold: float Range: -100 to 0

set_clipping(*clipping: float*) → None

```
# SCPI: SCENario:OUTPut:CLIPping
driver.scenario.output.set_clipping(clipping = 1.0)
```

Sets a maximum level to limit the dynamic range of the signal. Pulses at levels above this threshold are reduced (clipped) to the configured level.

param clipping
float Range: -100 to 20, Unit: dBm

set_format_py(*format_py: OutFormat*) → None

```
# SCPI: SCENario:OUTPut:FORMat
driver.scenario.output.set_format_py(format_py = enums.OutFormat.ESEQencing)
```

Sets the type of the generated waveform file.

param format_py
WV| MSW

set_frequency(*frequency: float*) → None

```
# SCPI: SCENario:OUTPut:FREQuency
driver.scenario.output.set_frequency(frequency = 1.0)
```

Sets the carrier RF frequency of the generated signal.

param frequency
float Range: 1000 to 1e+11

set_level(*level: float*) → None

```
# SCPI: SCENario:OUTPut:LEVel
driver.scenario.output.set_level(level = 1.0)
```

Sets the reference level used by the calculation of the pulse envelope.

param level
float Range: -130 to 30

set_mt_mode(*mt_mode: AutoManualMode*) → None

```
# SCPI: SCENario:OUTPut:MTMode
driver.scenario.output.set_mt_mode(mt_mode = enums.AutoManualMode.AUTO)
```

If multithreading is enabled with method RsPulseSeq.Scenario.Output.multithread, sets the mode to use for multithreading.

param mt_mode
AUTO| MANual

set_mt_threads(*mt_threads: float*) → None

```
# SCPI: SCENario:OUTPut:MTTHreads
driver.scenario.output.set_mt_threads(mt_threads = 1.0)
```

In manual mode, sets the required number of threads for the signal calculation.

param mt_threads
float Range: 0 to 1000

set_multithread(*multithread: bool*) → None

```
# SCPI: SCENario:OUTPut:MULTithread
driver.scenario.output.set_multithread(multithread = False)
```

Enable to optmize the calculation speed.

param multithread
ON| OFF| 1| 0

set_path(*path: str*) → None

```
# SCPI: SCENario:OUTPut:PATH
driver.scenario.output.set_path(path = 'abc')
```

Sets the directory the generated waveform is stored in.

param path
string File path

set_run_mode(*run_mode: RepeatMode*) → None

```
# SCPI: SCENario:OUTPut:RUNMode
driver.scenario.output.set_run_mode(run_mode = enums.RepeatMode.CONTinuous)
```

Defines the way the generated signal is processed.

param run_mode
CONTinuous| SINGle

set_target(*target: TargetOut*) → None

```
# SCPI: SCENario:OUTPut:TARGet
driver.scenario.output.set_target(target = enums.TargetOut.FILE)
```

Defines whether the software creates an ARB file or transfers the generated waveform to a connected physical generator. To assign a generator, use the command method RsPulseSeq.Scenario.Generator.value. To set the name and the directory the ARB file is stored in, use the command method RsPulseSeq.Scenario.Output.path.

param target
INSTrument| FILE

set_threshold(*threshold: float*) → None

```
# SCPI: SCENario:OUTPut:THReshold
driver.scenario.output.set_threshold(threshold = 1.0)
```

Sets a threshold. Pulses at levels below this threshold are omitted.

param threshold
float Range: -100 to 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.output.clone()
```

Subgroups

5.26.13.1 Arb

class ArbCls

Arb commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.output.arb.clone()
```

Subgroups

5.26.13.1.1 Details

SCPI Commands :

```
SCENario:OUTPut:ARB:DETails:ALBS
SCENario:OUTPut:ARB:DETails:TRUNcate
```

class DetailsCls

Details commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_albs() → bool

```
# SCPI: SCENario:OUTPut:ARB:DETails:ALBS
value: bool = driver.scenario.output.arb.details.get_albs()
```

Enables you to calculate the antenna attenuation for each sample. Otherwise a lookup at center position is used.

return
albs: ON| OFF| 1| 0

get_truncate() → bool

```
# SCPI: SCENario:OUTPut:ARB:DETails:TRUNcate
value: bool = driver.scenario.output.arb.details.get_truncate()
```

Enables signal truncation. If enabled, the calculation process allows truncated signals at the end of the signal duration, for example, only a fraction of a pulse if the time ends within the last pulse.

return
truncate: ON| OFF| 1| 0

set_albs(*albs: bool*) → None

```
# SCPI: SCENario:OUTPut:ARB:DETails:ALBS
driver.scenario.output.arb.details.set_albs(albs = False)
```

Enables you to calculate the antenna attenuation for each sample. Otherwise a lookup at center position is used.

param albs
ON| OFF| 1| 0

set_truncate(*truncate: bool*) → None

```
# SCPI: SCENario:OUTPut:ARB:DETails:TRUNcate
driver.scenario.output.arb.details.set_truncate(truncate = False)
```

Enables signal truncation. If enabled, the calculation process allows truncated signals at the end of the signal duration, for example, only a fraction of a pulse if the time ends within the last pulse.

param truncate
ON| OFF| 1| 0

5.26.13.2 Clock

SCPI Commands :

```
SCENario:OUTPut:CLOCk:MODE
SCENario:OUTPut:CLOCk:USER
```

class ClockCls

Clock commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_mode() → AutoManualMode

```
# SCPI: SCENario:OUTPut:CLOCk:MODE
value: enums.AutoManualMode = driver.scenario.output.clock.get_mode()
```

Sets the clock mode.

return
mode: AUTO| MANual AUTO Clock rate is retrieved from the generated waveform. MANual Clock rate is user-defined

get_user() → float

```
# SCPI: SCENario:OUTPut:CLOCk:USER
value: float = driver.scenario.output.clock.get_user()
```

Sets a user defined clock rate.

return
user: float Range: 1 to 2.4e+09

set_mode(*mode: AutoManualMode*) → None

```
# SCPI: SCENario:OUTPut:CLOCk:MODE
driver.scenario.output.clock.set_mode(mode = enums.AutoManualMode.AUTO)
```

Sets the clock mode.

param mode
AUTO| MANual AUTO Clock rate is retrieved from the generated waveform. MANual Clock rate is user-defined

set_user(*user: float*) → None

```
# SCPI: SCENario:OUTPut:CLOCk:USER
driver.scenario.output.clock.set_user(user = 1.0)
```

Sets a user defined clock rate.

param user
float Range: 1 to 2.4e+09

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.output.clock.clone()
```

Subgroups

5.26.13.2.1 Auto

SCPI Commands :

```
SCENario:OUTPut:CLOCk:AUTO:BORDer
SCENario:OUTPut:CLOCk:AUTO:OVERsampling
```

class AutoCls

Auto commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_border() → float

```
# SCPI: SCENario:OUTPut:CLOCk:AUTO:BORDer
value: float = driver.scenario.output.clock.auto.get_border()
```

Sets the minimum clock rate.

return
border: float Range: 1000 to 1e+08

get_oversampling() → float

```
# SCPI: SCENario:OUTPut:CLOCk:AUTO:OVERsampling
value: float = driver.scenario.output.clock.auto.get_oversampling()
```

Sets the minimum oversampling factor.

return
oversampling: float Range: 1 to 1000

set_border(*border: float*) → None

```
# SCPI: SCENario:OUTPut:CLOCk:AUTO:BORDer
driver.scenario.output.clock.auto.set_border(border = 1.0)
```

Sets the minimum clock rate.

param border
float Range: 1000 to 1e+08

set_oversampling(*oversampling: float*) → None

```
# SCPI: SCENario:OUTPut:CLOCk:AUTO:OVERsampling
driver.scenario.output.clock.auto.set_oversampling(oversampling = 1.0)
```

Sets the minimum oversampling factor.

param oversampling
float Range: 1 to 1000

5.26.13.3 Duration

SCPI Commands :

```
SCENario:OUTPut:DURation:AUTO
SCENario:OUTPut:DURation:MODE
SCENario:OUTPut:DURation:TIME
```

class DurationCls

Duration commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_auto() → float

```
# SCPI: SCENario:OUTPut:DURation:AUTO
value: float = driver.scenario.output.duration.get_auto()
```

Requires SCENario:OUTPut:DURation:MODE AUTO. Queries the value of the automatically determined signal duration.

return
auto: float Range: 1e-06 to 1.8432e+06

get_mode() → AutoManualMode

```
# SCPI: SCENario:OUTPut:DURation:MODE
value: enums.AutoManualMode = driver.scenario.output.duration.get_mode()
```

Sets how the waveform duration is defined.

return
mode: AUTO| MANual AUTO Sets the simulation time to maximum of sequence, scan or movement duration. MANual Sets the simulation time to a fixed value.

get_time() → float

```
# SCPI: SCENario:OUTPut:DURation:TIME
value: float = driver.scenario.output.duration.get_time()
```

Sets the duration of the generated waveform.

return
time: float Range: 1e-06 to 1.8432e+06 Simulation time longer than 7200s requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

set_mode(*mode: AutoManualMode*) → None

```
# SCPI: SCENario:OUTPut:DURation:MODE
driver.scenario.output.duration.set_mode(mode = enums.AutoManualMode.AUTO)
```

Sets how the waveform duration is defined.

param mode
AUTO| MANual AUTO Sets the simulation time to maximum of sequence, scan or movement duration. MANual Sets the simulation time to a fixed value.

set_time(*time: float*) → None

```
# SCPI: SCENario:OUTPut:DURation:TIME
driver.scenario.output.duration.set_time(time = 1.0)
```

Sets the duration of the generated waveform.

param time
float Range: 1e-06 to 1.8432e+06 Simulation time longer than 7200s requires R&S SMW with firmware version 5.xx.xxx and higher. To query the installed firmware version of the selected instrument, use the command method RsPulseSeq.Instrument.firmware.

5.26.13.4 Marker

SCPI Commands :

```
SCENario:OUTPut:MARKer:ENABle
SCENario:OUTPut:MARKer:FLAGs
```

class MarkerCls

Marker commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SCENario:OUTPut:MARKer:ENABle
value: bool = driver.scenario.output.marker.get_enable()
```

Enables that markers are considered by the generation of the output waveform file.

return
enable: ON| OFF| 1| 0

get_flags() → float

```
# SCPI: SCENario:OUTPut:MARKer:FLAGs
value: float = driver.scenario.output.marker.get_flags()
```

Enables up to four markers.

return
flags: int Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:OUTPut:MARKer:ENABle
driver.scenario.output.marker.set_enable(enable = False)
```

Enables that markers are considered by the generation of the output waveform file.

param enable
ON| OFF| 1| 0

set_flags(*flags: float*) → None

```
# SCPI: SCENario:OUTPut:MARKer:FLAGs
driver.scenario.output.marker.set_flags(flags = 1.0)
```

Enables up to four markers.

param flags
int Binary value, where: M1 = 1 M1 = 2 M1 = 4 M1 = 8 Range: 0 to 15

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.output.marker.clone()
```

Subgroups

5.26.13.4.1 Scenario

SCPI Commands :

```
SCENario:OUTPut:MARKer:SCENario:DURation
SCENario:OUTPut:MARKer:SCENario:ENABle
```

class ScenarioCls

Scenario commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_duration() → float

```
# SCPI: SCENario:OUTPut:MARKer:SCENario:DURation
value: float = driver.scenario.output.marker.scenario.get_duration()
```

Sets the duration of the scenario marker.

return
duration: float Range: 0 to 1, Unit: sec

get_enable() → bool

```
# SCPI: SCENario:OUTPut:MARKer:SCENario:ENABle
value: bool = driver.scenario.output.marker.scenario.get_enable()
```

Enables an additional marker, that is held high from the scenario start until the duration, selected with the command method RsPulseSeq.Scenario.Output.Marker.Scenario.duration.

return
enable: ON| OFF| 1| 0

set_duration(*duration: float*) → None

```
# SCPI: SCENario:OUTPut:MARKer:SCENario:DURation
driver.scenario.output.marker.scenario.set_duration(duration = 1.0)
```

Sets the duration of the scenario marker.

param duration
float Range: 0 to 1, Unit: sec

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:OUTPut:MARKer:SCENario:ENABle
driver.scenario.output.marker.scenario.set_enable(enable = False)
```

Enables an additional marker, that is held high from the scenario start until the duration, selected with the command method RsPulseSeq.Scenario.Output.Marker.Scenario.duration.

param enable
ON| OFF| 1| 0

5.26.13.5 Recall

SCPI Command :

```
SCENario:OUTPut:RECall:ENABle
```

class RecallCls

Recall commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SCENario:OUTPut:RECall:ENABle
value: bool = driver.scenario.output.recall.get_enable()
```

Stores the current signal generator configuration as a save/recall file.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:OUTPut:RECall:ENABle
driver.scenario.output.recall.set_enable(enable = False)
```

Stores the current signal generator configuration as a save/recall file.

param enable
ON| OFF| 1| 0

5.26.13.6 Reset

SCPI Command :

```
SCENario:OUTPut:RESet:ENABle
```

class ResetCls

Reset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SCENario:OUTPut:RESet:ENABle
value: bool = driver.scenario.output.reset.get_enable()
```

Restarts the connected instrument on scenario start.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:OUTPut:RESet:ENABle
driver.scenario.output.reset.set_enable(enable = False)
```

Restarts the connected instrument on scenario start.

param enable
ON| OFF| 1| 0

5.26.13.7 Supress

SCPI Command :

```
SCENario:OUTPut:SUPRess:ENABle
```

class SupressCls

Supress commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SCENario:OUTPut:SUPRess:ENABle
value: bool = driver.scenario.output.supress.get_enable()
```

Enable to prevent waveform recalculation if the RF frequency is changed.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:OUTPut:SUPRess:ENABle
driver.scenario.output.supress.set_enable(enable = False)
```

Enable to prevent waveform recalculation if the RF frequency is changed.

param enable
ON| OFF| 1| 0

5.26.14 Pdw

SCPI Commands :

```
SCENario:PDW:ENABle
SCENario:PDW:HOST
SCENario:PDW:PATH
SCENario:PDW:TEMPlate
SCENario:PDW:TYPE
```

class PdwCls

Pdw commands group definition. 16 total commands, 2 Subgroups, 5 group commands

get_enable() → bool

```
# SCPI: SCENario:PDW:ENABle
value: bool = driver.scenario.pdw.get_enable()
```

Enables generation of Pulse Descripter Word (PDW) reports.

return
enable: ON| OFF| 1| 0

get_host() → str

```
# SCPI: SCENario:PDW:HOST
value: str = driver.scenario.pdw.get_host()
```

Select a connected signal generator, with all options required for the scenario, to allow reporting.

return
host: string

get_path() → str

```
# SCPI: SCENario:PDW:PATH
value: str = driver.scenario.pdw.get_path()
```

Sets the target directory in that the generated report files are stored.

return
path: string

get_template() → str

```
# SCPI: SCENario:PDW:TEMPlate
value: str = driver.scenario.pdw.get_template()
```

Edits the selected template.

return
template: string

get_type_py() → PwdType

```
# SCPI: SCENario:PDW:TYPE
value: enums.PwdType = driver.scenario.pdw.get_type_py()
```

Sets the template used be the reporting function.

return
type_py: DEFault| TEMPlate| PLUGin| AMMos

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:PDW:ENABle
driver.scenario.pdw.set_enable(enable = False)
```

Enables generation of Pulse Descripter Word (PDW) reports.

param enable
ON| OFF| 1| 0

set_host(*host: str*) → None

```
# SCPI: SCENario:PDW:HOST
driver.scenario.pdw.set_host(host = 'abc')
```

Select a connected signal generator, with all options required for the scenario, to allow reporting.

param host
string

set_path(*path: str*) → None

```
# SCPI: SCENario:PDW:PATH
driver.scenario.pdw.set_path(path = 'abc')
```

Sets the target directory in that the generated report files are stored.

param path
string

set_template(*template: str*) → None

```
# SCPI: SCENario:PDW:TEMPlate
driver.scenario.pdw.set_template(template = 'abc')
```

Edits the selected template.

param template
string

set_type_py(*type_py: PwdType*) → None

```
# SCPI: SCENario:PDW:TYPE
driver.scenario.pdw.set_type_py(type_py = enums.PwdType.AMMos)
```

Sets the template used be the reporting function.

param type_py
DEFault| TEMPlate| PLUGin| AMMos

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.pdw.clone()
```

Subgroups

5.26.14.1 AmMos

SCPI Commands :

```
SCENario:PDW:AMMos:AZIMuth
SCENario:PDW:AMMos:FRAMe
SCENario:PDW:AMMos:PPDW
```

class AmMosCls

AmMos commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_azimuth() → Azimuth

```
# SCPI: SCENario:PDW:AMMos:AZIMuth
value: enums.Azimuth = driver.scenario.pdw.amMos.get_azimuth()
```

For method RsPulseSeq.Scenario.Pdw.typePyAMMos, defines whether the angle of the Rx antenna or the bearing is reported.

return
azimuth: RX| BEARing

get_frame() → float

```
# SCPI: SCENario:PDW:AMMos:FRAMe
value: float = driver.scenario.pdw.amMos.get_frame()
```

Sets the frame length.

return
frame: float Range: 50 to 500

get_ppdw() → bool

```
# SCPI: SCENario:PDW:AMMos:PPDW
value: bool = driver.scenario.pdw.amMos.get_ppdw()
```

If enabled, the format of the AMMOS file is set to PPDW. Otherwise PDW is assumed.

return
ppdw: ON| OFF| 1| 0

set_azimuth(*azimuth: Azimuth*) → None

```
# SCPI: SCENario:PDW:AMMos:AZIMuth
driver.scenario.pdw.amMos.set_azimuth(azimuth = enums.Azimuth.BEARing)
```

For method RsPulseSeq.Scenario.Pdw.typePyAMMos, defines whether the angle of the Rx antenna or the bearing is reported.

param azimuth
RX| BEARing

set_frame(*frame: float*) → None

```
# SCPI: SCENario:PDW:AMMos:FRAMe
driver.scenario.pdw.amMos.set_frame(frame = 1.0)
```

Sets the frame length.

param frame
float Range: 50 to 500

set_ppdw(*ppdw: bool*) → None

```
# SCPI: SCENario:PDW:AMMos:PPDW
driver.scenario.pdw.amMos.set_ppdw(ppdw = False)
```

If enabled, the format of the AMMOS file is set to PPDW. Otherwise PDW is assumed.

param ppdw
ON| OFF| 1| 0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.pdw.amMos.clone()
```

Subgroups

5.26.14.1.1 Utime

SCPI Commands :

```
SCENario:PDW:AMMos:UTIMe:ENABle
SCENario:PDW:AMMos:UTIMe:ISO
```

class UtimeCls

Utime commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SCENario:PDW:AMMos:UTIMe:ENABle
value: bool = driver.scenario.pdw.amMos.utime.get_enable()
```

Defines how the report start time is set.

return
enable: ON| OFF| 1| 0 0 The reporting start time is time at that the scenario calculation starts. 1 The reporting starts at user-defined moment, set with the command method RsPulseSeq.Scenario.Pdw.AmMos.Utime.iso.

get_iso() → str

```
# SCPI: SCENario:PDW:AMMos:UTIMe:ISO
value: str = driver.scenario.pdw.amMos.utime.get_iso()
```

Sets the reporting start time, if method RsPulseSeq.Scenario.Pdw.AmMos.Utime.enable1.

return
iso: 'YYYY-Month-DDTHH:MM:SS'

set_enable(*enable: bool*) → None

```
# SCPI: SCENario:PDW:AMMos:UTIMe:ENABle
driver.scenario.pdw.amMos.utime.set_enable(enable = False)
```

Defines how the report start time is set.

param enable
ON| OFF| 1| 0 0 The reporting start time is time at that the scenario calculation starts. 1 The reporting starts at user-defined moment, set with the command method RsPulseSeq.Scenario.Pdw.AmMos.Utime.iso.

set_iso(*iso: str*) → None

```
# SCPI: SCENario:PDW:AMMos:UTIMe:ISO
driver.scenario.pdw.amMos.utime.set_iso(iso = 'abc')
```

Sets the reporting start time, if method RsPulseSeq.Scenario.Pdw.AmMos.Utime.enable1.

param iso
'YYYY-Month-DDTHH:MM:SS'

5.26.14.2 Plugin

SCPI Command :

```
SCENario:PDW:PLUGin:NAME
```

class PluginCls

Plugin commands group definition. 6 total commands, 1 Subgroups, 1 group commands

get_name() → str

```
# SCPI: SCENario:PDW:PLUGin:NAME
value: str = driver.scenario.pdw.plugin.get_name()
```

Selects and loads a reporting template. This template must exist in the 'Plugin' library. To query a list of available plugins, use the command method RsPulseSeq.Plugin.catalog.

return
name: string

set_name(*name: str*) → None

```
# SCPI: SCENario:PDW:PLUGin:NAME
driver.scenario.pdw.plugin.set_name(name = 'abc')
```

Selects and loads a reporting template. This template must exist in the 'Plugin' library. To query a list of available plugins, use the command method RsPulseSeq.Plugin.catalog.

param name
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.pdw.plugin.clone()
```

Subgroups

5.26.14.2.1 Variable

SCPI Commands :

```
SCENario:PDW:PLUGin:VARiable:CATalog
SCENario:PDW:PLUGin:VARiable:RESet
SCENario:PDW:PLUGin:VARiable:VALue
```

class VariableCls

Variable commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_catalog() → str

```
# SCPI: SCENario:PDW:PLUGin:VARiable:CATalog
value: str = driver.scenario.pdw.plugin.variable.get_catalog()
```

Queries the variables used in the plugin.

return
catalog: string

get_value() → str

```
# SCPI: SCENario:PDW:PLUGin:VARiable:VALue
value: str = driver.scenario.pdw.plugin.variable.get_value()
```

Sets the values of the selected variable.

return
value: string

reset() → None

```
# SCPI: SCENario:PDW:PLUGin:VARiable:RESet
driver.scenario.pdw.plugin.variable.reset()
```

Resets the variable values to the defaults.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:PDW:PLUGin:VARiable:RESet
driver.scenario.pdw.plugin.variable.reset_with_opc()
```

Resets the variable values to the defaults.

Same as reset, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*value: str*) → None

```
# SCPI: SCENario:PDW:PLUGin:VARiable:VALue
driver.scenario.pdw.plugin.variable.set_value(value = 'abc')
```

Sets the values of the selected variable.

param value
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.pdw.plugin.variable.clone()
```

Subgroups

5.26.14.2.1.1 Select

SCPI Commands :

```
SCENario:PDW:PLUGin:VARiable:SELect:ID
SCENario:PDW:PLUGin:VARiable:SELect
```

class SelectCls

Select commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_id() → float

```
# SCPI: SCENario:PDW:PLUGin:VARiable:SELect:ID
value: float = driver.scenario.pdw.plugin.variable.select.get_id()
```

Selects a plugin variable ID.

return
idn: float

get_value() → str

```
# SCPI: SCENario:PDW:PLUGin:VARiable:SELect
value: str = driver.scenario.pdw.plugin.variable.select.get_value()
```

Selects a plugin variable.

return
select: string

set_id(*idn: float*) → None

```
# SCPI: SCENario:PDW:PLUGin:VARiable:SELect:ID
driver.scenario.pdw.plugin.variable.select.set_id(idn = 1.0)
```

Selects a plugin variable ID.

param idn
float

set_value(*select: str*) → None

```
# SCPI: SCENario:PDW:PLUGin:VARiable:SELect
driver.scenario.pdw.plugin.variable.select.set_value(select = 'abc')
```

Selects a plugin variable.

param select
string

5.26.15 Sequence

SCPI Commands :

```
SCENario:SEQuence:CLEar
SCENario:SEQuence
```

class SequenceCls

Sequence commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: SCENario:SEQuence:CLEar
driver.scenario.sequence.clear()
```

No command help available

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:SEQuence:CLEar
driver.scenario.sequence.clear_with_opc()
```

No command help available

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_value() → str

```
# SCPI: SCENario:SEQuence
value: str = driver.scenario.sequence.get_value()
```

Assigns a pulse sequence, see method RsPulseSeq.Sequence.catalog.

return
sequence: string

set_value(*sequence: str*) → None

```
# SCPI: SCENario:SEQuence
driver.scenario.sequence.set_value(sequence = 'abc')
```

Assigns a pulse sequence, see method RsPulseSeq.Sequence.catalog.

param sequence
string

5.26.16 Trigger

SCPI Command :

```
SCENario:TRIGger
```

class TriggerCls

Trigger commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SCENario:TRIGger
driver.scenario.trigger.set()
```

Triggers the scenario, if separate trigger for scenario start is enabled. See method RsPulseSeq.Program.Scenario.Xtrg. enable.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:TRIGger
driver.scenario.trigger.set_with_opc()
```

Triggers the scenario, if separate trigger for scenario start is enabled. See method RsPulseSeq.Program.Scenario.Xtrg. enable.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.26.17 Volatile

SCPI Command :

```
SCENario:VOLatile:SEL
```

class VolatileCls

Volatile commands group definition. 6 total commands, 1 Subgroups, 1 group commands

get_sel() → float

```
# SCPI: SCENario:VOLatile:SEL
value: float = driver.scenario.volatile.get_sel()
```

If several files are created, select the one to be visualized.

return
sel: float Subsequent number, indicating the files in the volatile memory.

set_sel(*sel: float*) → None

```
# SCPI: SCENario:VOLatile:SEL
driver.scenario.volatile.set_sel(sel = 1.0)
```

If several files are created, select the one to be visualized.

param sel
float Subsequent number, indicating the files in the volatile memory.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.volatile.clone()
```

Subgroups

5.26.17.1 View

SCPI Commands :

```
SCENario:VOLatile:VIEW
SCENario:VOLatile:VIEW:XMODe
SCENario:VOLatile:VIEW:YMODe
```

class ViewCls

View commands group definition. 5 total commands, 1 Subgroups, 3 group commands

set() → None

```
# SCPI: SCENario:VOLatile:VIEW
driver.scenario.volatile.view.set()
```

If a waveform exists in the volatile memory, opens the 'Waveform Viewer' and displays this waveform.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SCENario:VOLatile:VIEW
driver.scenario.volatile.view.set_with_opc()
```

If a waveform exists in the volatile memory, opens the 'Waveform Viewer' and displays this waveform.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_xmode(*xmode: ViewXode*) → None

```
# SCPI: SCENario:VOLatile:VIEW:XMODe
driver.scenario.volatile.view.set_xmode(xmode = enums.ViewXode.SAMPles)
```

Sets the units (time or samples) used on the x axis.

param xmode
SAMPles| TIME

set_ymode(*ymode: Ymode*) → None

```
# SCPI: SCENario:VOLatile:VIEW:YMODe
driver.scenario.volatile.view.set_ymode(ymode = enums.Ymode.FREQuency)
```

Sets the view mode.

param ymode
IQ| MAGDb| MAGW| MAGV| PHASe| FREQuency| PAV

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scenario.volatile.view.clone()
```

Subgroups

5.26.17.1.1 Zoom

SCPI Commands :

```
SCENario:VOLatile:VIEW:ZOOM:POINt
SCENario:VOLatile:VIEW:ZOOM:RANGe
```

class ZoomCls

Zoom commands group definition. 2 total commands, 0 Subgroups, 2 group commands

set_point(*point: float*) → None

```
# SCPI: SCENario:VOLatile:VIEW:ZOOM:POINt
driver.scenario.volatile.view.zoom.set_point(point = 1.0)
```

Sets center point of the displayed area.

param point
float Always related to time Unit: s

set_range(*range_py: float*) → None

```
# SCPI: SCENario:VOLatile:VIEW:ZOOM:RANGe
driver.scenario.volatile.view.zoom.set_range(range_py = 1.0)
```

Sets the displayed waveform part as a range around the selected center point, set with the command method RsPulseSeq. Scenario.Volatile.View.Zoom.point.

param range_py
float Expressed as a time span (units can be omitted) or as number of samples

5.27 Script

SCPI Command :

```
SCRipt:ADD
```

class ScriptCls

Script commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set_add(*add: str*) → None

```
# SCPI: SCRipt:ADD
driver.script.set_add(add = 'abc')
```

No command help available

param add
No help available

5.28 Sequence

SCPI Commands :

```
SEQuence:CATalog
SEQuence:COMMent
SEQuence:CREate
SEQuence:NAME
SEQuence:REMove
SEQuence:SELect
SEQuence:TYPE
```

class SequenceCls

Sequence commands group definition. 65 total commands, 3 Subgroups, 7 group commands

get_catalog() → str

```
# SCPI: SEQuence:CATalog
value: str = driver.sequence.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: SEQuence:COMMent
value: str = driver.sequence.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_name() → str

```
# SCPI: SEQuence:NAME
value: str = driver.sequence.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: SEQuence:SELect
value: str = driver.sequence.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

get_type_py() → SequenceType

```
# SCPI: SEQuence:TYPE
value: enums.SequenceType = driver.sequence.get_type_py()
```

Sets the sequence type.

return
type_py: PULSe | | WAVeform

set_comment(*comment: str*) → None

```
# SCPI: SEQuence:COMMent
driver.sequence.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: SEQuence:CREate
driver.sequence.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: SEQuence:NAME
driver.sequence.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: SEQuence:REMove
driver.sequence.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: SEQuence:SELect
driver.sequence.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_type_py(*type_py: SequenceType*) → None

```
# SCPI: SEQuence:TYPE
driver.sequence.set_type_py(type_py = enums.SequenceType.PULSe)
```

Sets the sequence type.

param type_py
PULSe | | WAVeform

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sequence.clone()
```

Subgroups

5.28.1 Item

SCPI Commands :

```
SEQuence:ITEM:CLEar
SEQuence:ITEM:COUNt
SEQuence:ITEM:DELete
SEQuence:ITEM:INDent
SEQuence:ITEM:PDELay
SEQuence:ITEM:PRF
SEQuence:ITEM:PRI
SEQuence:ITEM:PULSe
SEQuence:ITEM:SELect
SEQuence:ITEM:TYPE
SEQuence:ITEM:WAVeform
```

class ItemCls

Item commands group definition. 56 total commands, 10 Subgroups, 11 group commands

clear() → None

```
# SCPI: SEQuence:ITEM:CLEar
driver.sequence.item.clear()
```

Deletes all items from the list or the table.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SEQuence:ITEM:CLEar
driver.sequence.item.clear_with_opc()
```

Deletes all items from the list or the table.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

delete(*delete: float*) → None

```
# SCPI: SEQuence:ITEM:DELete
driver.sequence.item.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SEQuence:ITEM:COUNt
value: float = driver.sequence.item.get_count()
```

Queries the number of existing items.

return
count: integer

get_indent() → float

```
# SCPI: SEQuence:ITEM:INDent
value: float = driver.sequence.item.get_indent()
```

Indents the selected item rows to include it, for example, in a loop.

return
indent: float Range: 0 to 5

get_pdelay() → float

```
# SCPI: SEQuence:ITEM:PDELay
value: float = driver.sequence.item.get_pdelay()
```

Enables a start delay.

return
pdelay: float Range: 0 to 1e+09, Unit: sec

get_prf() → float

```
# SCPI: SEQuence:ITEM:PRF
value: float = driver.sequence.item.get_prf()
```

Sets the pulse repetition interval (PRI) or the pulse repetition frequency (PRF) .

return
prf: No help available

get_pri() → float

```
# SCPI: SEQuence:ITEM:PRI
value: float = driver.sequence.item.get_pri()
```

Sets the pulse repetition interval (PRI) or the pulse repetition frequency (PRF) .

return
pri: float Range: 0 to 1e+09

get_pulse() → str

```
# SCPI: SEQuence:ITEM:PULSe
value: str = driver.sequence.item.get_pulse()
```

Assigns a pulse or a waveform to the selected item. Use the commands method RsPulseSeq.Pulse. catalog and method RsPulseSeq.Waveform.catalog to querry the available pulses and waveforms.

return
pulse: string Pulse name

get_select() → float

```
# SCPI: SEQuence:ITEM:SELect
value: float = driver.sequence.item.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

get_type_py() → ItemType

```
# SCPI: SEQuence:ITEM:TYPE
value: enums.ItemType = driver.sequence.item.get_type_py()
```

Sets the content type of the selected item.

return
type_py: PULSe| FILLer| LOOP | | OVL| SUBSequence| WAVeform

get_waveform() → str

```
# SCPI: SEQuence:ITEM:WAVeform
value: str = driver.sequence.item.get_waveform()
```

Assigns a pulse or a waveform to the selected item. Use the commands method RsPulseSeq.Pulse. catalog and method RsPulseSeq.Waveform.catalog to querry the available pulses and waveforms.

return
waveform: No help available

set_indent(*indent: float*) → None

```
# SCPI: SEQuence:ITEM:INDent
driver.sequence.item.set_indent(indent = 1.0)
```

Indents the selected item rows to include it, for example, in a loop.

param indent
float Range: 0 to 5

set_pdelay(*pdelay: float*) → None

```
# SCPI: SEQuence:ITEM:PDELay
driver.sequence.item.set_pdelay(pdelay = 1.0)
```

Enables a start delay.

param pdelay
float Range: 0 to 1e+09, Unit: sec

set_prf(*prf: float*) → None

```
# SCPI: SEQuence:ITEM:PRF
driver.sequence.item.set_prf(prf = 1.0)
```

Sets the pulse repetition interval (PRI) or the pulse repetition frequency (PRF) .

param prf
float Range: 0 to 1e+09

set_pri(*pri: float*) → None

```
# SCPI: SEQuence:ITEM:PRI
driver.sequence.item.set_pri(pri = 1.0)
```

Sets the pulse repetition interval (PRI) or the pulse repetition frequency (PRF) .

param pri
float Range: 0 to 1e+09

set_pulse(*pulse: str*) → None

```
# SCPI: SEQuence:ITEM:PULSe
driver.sequence.item.set_pulse(pulse = 'abc')
```

Assigns a pulse or a waveform to the selected item. Use the commands method RsPulseSeq.Pulse. catalog and method RsPulseSeq.Waveform.catalog to querry the available pulses and waveforms.

param pulse
string Pulse name

set_select(*select: float*) → None

```
# SCPI: SEQuence:ITEM:SELect
driver.sequence.item.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_type_py(*type_py: ItemType*) → None

```
# SCPI: SEQuence:ITEM:TYPE
driver.sequence.item.set_type_py(type_py = enums.ItemType.FILLer)
```

Sets the content type of the selected item.

param type_py
PULSe| FILLer| LOOP | | OVL| SUBSequence| WAVeform

set_waveform(*waveform: str*) → None

```
# SCPI: SEQuence:ITEM:WAVeform
driver.sequence.item.set_waveform(waveform = 'abc')
```

Assigns a pulse or a waveform to the selected item. Use the commands method RsPulseSeq.Pulse. catalog and method RsPulseSeq.Waveform.catalog to querry the available pulses and waveforms.

param waveform
string Pulse name

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sequence.item.clone()
```

Subgroups

5.28.1.1 Add

SCPI Command :

```
SEQuence:ITEM:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SEQuence:ITEM:ADD
driver.sequence.item.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SEQuence:ITEM:ADD
driver.sequence.item.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.28.1.2 Filler

SCPI Commands :

```
SEQuence:ITEM:FILLer:MODE
SEQuence:ITEM:FILLer:SIGNal
```

class FillerCls

Filler commands group definition. 5 total commands, 1 Subgroups, 2 group commands

get_mode() → FillerMode

```
# SCPI: SEQuence:ITEM:FILLer:MODE
value: enums.FillerMode = driver.sequence.item.filler.get_mode()
```

Sets how the filler duration is determined.

return
mode: DURation| TSYNc

get_signal() → FillerSignal

```
# SCPI: SEQuence:ITEM:FILLer:SIGNal
value: enums.FillerSignal = driver.sequence.item.filler.get_signal()
```

Sets the signal type.

return
signal: BLANk | | CW| HOLD

set_mode(*mode: FillerMode*) → None

```
# SCPI: SEQuence:ITEM:FILLer:MODE
driver.sequence.item.filler.set_mode(mode = enums.FillerMode.DURation)
```

Sets how the filler duration is determined.

param mode
DURation| TSYNc

set_signal(*signal: FillerSignal*) → None

```
# SCPI: SEQuence:ITEM:FILLer:SIGNal
driver.sequence.item.filler.set_signal(signal = enums.FillerSignal.BLANk)
```

Sets the signal type.

param signal
BLANk | | CW| HOLD

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sequence.item.filler.clone()
```

Subgroups

5.28.1.2.1 Time

SCPI Commands :

```
SEQuence:ITEM:FILLer:TIME:EQUation
SEQuence:ITEM:FILLer:TIME:FIXed
SEQuence:ITEM:FILLer:TIME
```

class TimeCls

Time commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_equation() → str

```
# SCPI: SEQuence:ITEM:FILLer:TIME:EQUation
value: str = driver.sequence.item.filler.time.get_equation()
```

Sets the filler duration as an equation.

return
equation: string

get_fixed() → float

```
# SCPI: SEQuence:ITEM:FILLer:TIME:FIXed
value: float = driver.sequence.item.filler.time.get_fixed()
```

Sets the duration of the filler.

return
fixed: float Range: 0 to 1e+09, Unit: sec

get_value() → FillerTime

```
# SCPI: SEQuence:ITEM:FILLer:TIME
value: enums.FillerTime = driver.sequence.item.filler.time.get_value()
```

Defines the way the duration is defined.

return
time: FIXed| EQUation

set_equation(*equation: str*) → None

```
# SCPI: SEQuence:ITEM:FILLer:TIME:EQUation
driver.sequence.item.filler.time.set_equation(equation = 'abc')
```

Sets the filler duration as an equation.

param equation
string

set_fixed(*fixed: float*) → None

```
# SCPI: SEQuence:ITEM:FILLer:TIME:FIXed
driver.sequence.item.filler.time.set_fixed(fixed = 1.0)
```

Sets the duration of the filler.

param fixed
float Range: 0 to 1e+09, Unit: sec

set_value(*time: FillerTime*) → None

```
# SCPI: SEQuence:ITEM:FILLer:TIME
driver.sequence.item.filler.time.set_value(time = enums.FillerTime.EQUation)
```

Defines the way the duration is defined.

param time
FIXed| EQUation

5.28.1.3 Frequency

SCPI Command :

```
SEQuence:ITEM:FREQuency:OFFSet
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_offset() → float

```
# SCPI: SEQuence:ITEM:FREQuency:OFFSet
value: float = driver.sequence.item.frequency.get_offset()
```

Enables a frequency offset.

return
offset: float Range: -1e+09 to 1e+09, Unit: Hz

set_offset(*offset: float*) → None

```
# SCPI: SEQuence:ITEM:FREQuency:OFFSet
driver.sequence.item.frequency.set_offset(offset = 1.0)
```

Enables a frequency offset.

param offset
float Range: -1e+09 to 1e+09, Unit: Hz

5.28.1.4 Ipm

SCPI Commands :

```
SEQuence:ITEM:IPM:COUNt
SEQuence:ITEM:IPM:DELete
SEQuence:ITEM:IPM:EQUation
SEQuence:ITEM:IPM:MODE
SEQuence:ITEM:IPM:RESTart
SEQuence:ITEM:IPM:SELect
```

class IpmCls

Ipm commands group definition. 14 total commands, 4 Subgroups, 6 group commands

delete(*delete: float*) → None

```
# SCPI: SEQuence:ITEM:IPM:DELete
driver.sequence.item.ipm.delete(delete = 1.0)
```

Deletes the particular item.

param delete
float

get_count() → float

```
# SCPI: SEQuence:ITEM:IPM:COUNt
value: float = driver.sequence.item.ipm.get_count()
```

Queries the number of existing items.

return
count: integer

get_equation() → str

```
# SCPI: SEQuence:ITEM:IPM:EQUation
value: str = driver.sequence.item.ipm.get_equation()
```

Defines output value of the IPM mathematically.

return
equation: string

get_mode() → IpmMode

```
# SCPI: SEQuence:ITEM:IPM:MODE
value: enums.IpmMode = driver.sequence.item.ipm.get_mode()
```

Defines the way the variations are applied on repeating pulses.

return
mode: INDividual| SAME

get_restart() → bool

```
# SCPI: SEQuence:ITEM:IPM:RESTart
value: bool = driver.sequence.item.ipm.get_restart()
```

Restarts the IPM for this sequence line item.

return
restart: ON| OFF| 1| 0

get_select() → float

```
# SCPI: SEQuence:ITEM:IPM:SELect
value: float = driver.sequence.item.ipm.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_equation(*equation: str*) → None

```
# SCPI: SEQuence:ITEM:IPM:EQUation
driver.sequence.item.ipm.set_equation(equation = 'abc')
```

Defines output value of the IPM mathematically.

param equation
string

set_mode(*mode: IpmMode*) → None

```
# SCPI: SEQuence:ITEM:IPM:MODE
driver.sequence.item.ipm.set_mode(mode = enums.IpmMode.INDividual)
```

Defines the way the variations are applied on repeating pulses.

param mode
INDividual| SAME

set_restart(*restart: bool*) → None

```
# SCPI: SEQuence:ITEM:IPM:RESTart
driver.sequence.item.ipm.set_restart(restart = False)
```

Restarts the IPM for this sequence line item.

param restart
ON| OFF| 1| 0

set_select(*select: float*) → None

```
# SCPI: SEQuence:ITEM:IPM:SELect
driver.sequence.item.ipm.set_select(select = 1.0)
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sequence.item.ipm.clone()
```

Subgroups

5.28.1.4.1 Add

SCPI Command :

```
SEQuence:ITEM:IPM:ADD
```

class AddCls

Add commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SEQuence:ITEM:IPM:ADD
driver.sequence.item.ipm.add.set()
```

Appends new item.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SEQuence:ITEM:IPM:ADD
driver.sequence.item.ipm.add.set_with_opc()
```

Appends new item.

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.28.1.4.2 Random

SCPI Command :

```
SEQuence:ITEM:IPM:RANDom:RESet
```

class RandomCls

Random commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_reset() → bool

```
# SCPI: SEQuence:ITEM:IPM:RANDom:RESet
value: bool = driver.sequence.item.ipm.random.get_reset()
```

Resets the start seed of random generator.

return
reset: ON| OFF| 1| 0

set_reset(*reset: bool*) → None

```
# SCPI: SEQuence:ITEM:IPM:RANDom:RESet
driver.sequence.item.ipm.random.set_reset(reset = False)
```

Resets the start seed of random generator.

param reset
ON| OFF| 1| 0

5.28.1.4.3 Source

SCPI Commands :

```
SEQuence:ITEM:IPM:SOURce:TYPE
SEQuence:ITEM:IPM:SOURce:VARiable
SEQuence:ITEM:IPM:SOURce
```

class SourceCls

Source commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_type_py() → SourceType

```
# SCPI: SEQuence:ITEM:IPM:SOURce:TYPE
value: enums.SourceType = driver.sequence.item.ipm.source.get_type_py()
```

Sets whether the variation is defined as a profile or as a variable.

return
type_py: PROFile| VARiable

get_value() → str

```
# SCPI: SEQuence:ITEM:IPM:SOURce
value: str = driver.sequence.item.ipm.source.get_value()
```

Selects the profile source. Use the command method RsPulseSeq.Ipm.catalog to querry the existing profiles.

return
source: string

get_variable() → str

```
# SCPI: SEQuence:ITEM:IPM:SOURce:VARiable
value: str = driver.sequence.item.ipm.source.get_variable()
```

Sets the variable that defines the variation.

return
variable: string

set_type_py(*type_py: SourceType*) → None

```
# SCPI: SEQuence:ITEM:IPM:SOURce:TYPE
driver.sequence.item.ipm.source.set_type_py(type_py = enums.SourceType.PROFile)
```

Sets whether the variation is defined as a profile or as a variable.

param type_py
PROFile| VARiable

set_value(*source: str*) → None

```
# SCPI: SEQuence:ITEM:IPM:SOURce
driver.sequence.item.ipm.source.set_value(source = 'abc')
```

Selects the profile source. Use the command method RsPulseSeq.Ipm.catalog to querry the existing profiles.

param source
string

set_variable(*variable: str*) → None

```
# SCPI: SEQuence:ITEM:IPM:SOURce:VARiable
driver.sequence.item.ipm.source.set_variable(variable = 'abc')
```

Sets the variable that defines the variation.

param variable
string

5.28.1.4.4 Target

SCPI Commands :

```
SEQuence:ITEM:IPM:TARGet:PARameter
SEQuence:ITEM:IPM:TARGet:TYPE
SEQuence:ITEM:IPM:TARGet:VARiable
```

class TargetCls

Target commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_parameter() → TargetParam

```
# SCPI: SEQuence:ITEM:IPM:TARGet:PARameter
value: enums.TargetParam = driver.sequence.item.ipm.target.get_parameter()
```

Sets the pulse parameter to that the IPM variation is assigned.

return
parameter: LEVel| RLEVel| SRATe| FREQuency| PRI| WIDTh| FALL| AMFRequency| FMDeviation| DELay| FSKDeviation| PRF| FMFRequency| CDEViation| PHASe| RISE| AMDepth | DROop| RFRequency| OVERshoot

get_type_py() → TargetType

```
# SCPI: SEQuence:ITEM:IPM:TARGet:TYPE
value: enums.TargetType = driver.sequence.item.ipm.target.get_type_py()
```

Sets whether the profile is assigned to a parameter or to a variable.

return
type_py: PARameter| VARiable

get_variable() → str

```
# SCPI: SEQuence:ITEM:IPM:TARGet:VARiable
value: str = driver.sequence.item.ipm.target.get_variable()
```

Sets the variable to that the variation is assigned.

return
variable: string

set_parameter(*parameter: TargetParam*) → None

```
# SCPI: SEQuence:ITEM:IPM:TARGet:PARameter
driver.sequence.item.ipm.target.set_parameter(parameter = enums.TargetParam.
↪AMDepth)
```

Sets the pulse parameter to that the IPM variation is assigned.

param parameter
LEVel| RLEVel| SRATe| FREQuency| PRI| WIDTh| FALL| AMFRequency| FMDeviation| DELay| FSKDeviation| PRF| FMFRequency| CDEViation| PHASe| RISE| AMDepth | DROop| RFRequency| OVERshoot

set_type_py(*type_py: TargetType*) → None

```
# SCPI: SEQuence:ITEM:IPM:TARGet:TYPE
driver.sequence.item.ipm.target.set_type_py(type_py = enums.TargetType.
↪PARameter)
```

Sets whether the profile is assigned to a parameter or to a variable.

param type_py
PARameter| VARiable

set_variable(*variable: str*) → None

```
# SCPI: SEQuence:ITEM:IPM:TARGet:VARiable
driver.sequence.item.ipm.target.set_variable(variable = 'abc')
```

Sets the variable to that the variation is assigned.

param variable
string

5.28.1.5 Level

SCPI Command :

```
SEQuence:ITEM:LEVel:OFFSet
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_offset() → float

```
# SCPI: SEQuence:ITEM:LEVel:OFFSet
value: float = driver.sequence.item.level.get_offset()
```

Sets a level offset.

return
offset: float Range: -100 to 0, Unit: dB

set_offset(*offset: float*) → None

```
# SCPI: SEQuence:ITEM:LEVel:OFFSet
driver.sequence.item.level.set_offset(offset = 1.0)
```

Sets a level offset.

param offset
float Range: -100 to 0, Unit: dB

5.28.1.6 Loop

SCPI Commands :

```
SEQuence:ITEM:LOOP:TYPE
SEQuence:ITEM:LOOP:VARiable
```

class LoopCls

Loop commands group definition. 6 total commands, 1 Subgroups, 2 group commands

get_type_py() → LoopType

```
# SCPI: SEQuence:ITEM:LOOP:TYPE
value: enums.LoopType = driver.sequence.item.loop.get_type_py()
```

Sets how the loop repetition is defined.

return
type_py: FIXed| VARiable

get_variable() → str

```
# SCPI: SEQuence:ITEM:LOOP:VARiable
value: str = driver.sequence.item.loop.get_variable()
```

Sets a loop variable.

return
variable: string

set_type_py(*type_py: LoopType*) → None

```
# SCPI: SEQuence:ITEM:LOOP:TYPE
driver.sequence.item.loop.set_type_py(type_py = enums.LoopType.FIXed)
```

Sets how the loop repetition is defined.

param type_py
FIXed| VARiable

set_variable(*variable: str*) → None

```
# SCPI: SEQuence:ITEM:LOOP:VARiable
driver.sequence.item.loop.set_variable(variable = 'abc')
```

Sets a loop variable.

param variable
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sequence.item.loop.clone()
```

Subgroups

5.28.1.6.1 Count

SCPI Commands :

```
SEQuence:ITEM:LOOP:COUNt:FIXed
SEQuence:ITEM:LOOP:COUNt:MAXimum
SEQuence:ITEM:LOOP:COUNt:MINimum
SEQuence:ITEM:LOOP:COUNt:STEP
```

class CountCls

Count commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_fixed() → float

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:FIXed
value: float = driver.sequence.item.loop.count.get_fixed()
```

Sets the repetition number as a numeric value.

return
fixed: float Range: 1 to 65535

get_maximum() → float

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:MAXimum
value: float = driver.sequence.item.loop.count.get_maximum()
```

Sets the value range of the loop count.

return
maximum: float Range: 1 to 65535

get_minimum() → float

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:MINimum
value: float = driver.sequence.item.loop.count.get_minimum()
```

Sets the value range of the loop count.

return
minimum: No help available

get_step() → float

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:STEP
value: float = driver.sequence.item.loop.count.get_step()
```

Sets the loop count granularity.

return
step: float Range: 1 to 65535

set_fixed(*fixed: float*) → None

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:FIXed
driver.sequence.item.loop.count.set_fixed(fixed = 1.0)
```

Sets the repetition number as a numeric value.

param fixed
float Range: 1 to 65535

set_maximum(*maximum: float*) → None

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:MAXimum
driver.sequence.item.loop.count.set_maximum(maximum = 1.0)
```

Sets the value range of the loop count.

param maximum
float Range: 1 to 65535

set_minimum(*minimum: float*) → None

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:MINimum
driver.sequence.item.loop.count.set_minimum(minimum = 1.0)
```

Sets the value range of the loop count.

param minimum
float Range: 1 to 65535

set_step(*step: float*) → None

```
# SCPI: SEQuence:ITEM:LOOP:COUNt:STEP
driver.sequence.item.loop.count.set_step(step = 1.0)
```

Sets the loop count granularity.

param step
float Range: 1 to 65535

5.28.1.7 Marker

SCPI Commands :

```
SEQuence:ITEM:MARKer:ALL
SEQuence:ITEM:MARKer:FIRSt
SEQuence:ITEM:MARKer:LAST
```

class MarkerCls

Marker commands group definition. 7 total commands, 1 Subgroups, 3 group commands

get_all() → float

```
# SCPI: SEQuence:ITEM:MARKer:ALL
value: float = driver.sequence.item.marker.get_all()
```

Enables up to four markers of the corresponding type.

return
all_py: float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

get_first() → float

```
# SCPI: SEQuence:ITEM:MARKer:FIRSt
value: float = driver.sequence.item.marker.get_first()
```

Enables up to four markers of the corresponding type.

return
first: No help available

get_last() → float

```
# SCPI: SEQuence:ITEM:MARKer:LAST
value: float = driver.sequence.item.marker.get_last()
```

Enables up to four markers of the corresponding type.

return
last: No help available

set_all(*all_py: float*) → None

```
# SCPI: SEQuence:ITEM:MARKer:ALL
driver.sequence.item.marker.set_all(all_py = 1.0)
```

Enables up to four markers of the corresponding type.

param all_py
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_first(*first: float*) → None

```
# SCPI: SEQuence:ITEM:MARKer:FIRSt
driver.sequence.item.marker.set_first(first = 1.0)
```

Enables up to four markers of the corresponding type.

param first
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_last(*last: float*) → None

```
# SCPI: SEQuence:ITEM:MARKer:LAST
driver.sequence.item.marker.set_last(last = 1.0)
```

Enables up to four markers of the corresponding type.

param last
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sequence.item.marker.clone()
```

Subgroups

5.28.1.7.1 Condition

SCPI Commands :

```
SEQuence:ITEM:MARKer:CONDition:TYPE
SEQuence:ITEM:MARKer:CONDition:VALue
SEQuence:ITEM:MARKer:CONDition:VARiable
SEQuence:ITEM:MARKer:CONDition
```

class ConditionCls

Condition commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_string_value() → str

```
# SCPI: SEQuence:ITEM:MARKer:CONDition:VALue
value: str = driver.sequence.item.marker.condition.get_string_value()
```

Sets the numerical value used with the comparison.

return
value: string

get_type_py() → Condition

```
# SCPI: SEQuence:ITEM:MARKer:CONDition:TYPE
value: enums.Condition = driver.sequence.item.marker.condition.get_type_py()
```

Sets the sign in the logical condition.

return
type_py: SMALler| GREater| EQUal| NOTequal

get_value() → float

```
# SCPI: SEQuence:ITEM:MARKer:CONDition
value: float = driver.sequence.item.marker.condition.get_value()
```

Enables up to four markers of the corresponding type.

return
condition: No help available

get_variable() → str

```
# SCPI: SEQuence:ITEM:MARKer:CONDition:VARiable
value: str = driver.sequence.item.marker.condition.get_variable()
```

Defines the value that is compared with the fixed values set with the command method RsPulseSeq.Sequence.Item.Marker. Condition.stringValue.

return
variable: string

set_string_value(*value: str*) → None

```
# SCPI: SEQuence:ITEM:MARKer:CONDition:VALue
driver.sequence.item.marker.condition.set_string_value(value = 'abc')
```

Sets the numerical value used with the comparison.

param value
string

set_type_py(*type_py: Condition*) → None

```
# SCPI: SEQuence:ITEM:MARKer:CONDition:TYPE
driver.sequence.item.marker.condition.set_type_py(type_py = enums.Condition.
↪EQUal)
```

Sets the sign in the logical condition.

param type_py
SMALLer| GREater| EQUal| NOTequal

set_value(*condition: float*) → None

```
# SCPI: SEQuence:ITEM:MARKer:CONDition
driver.sequence.item.marker.condition.set_value(condition = 1.0)
```

Enables up to four markers of the corresponding type.

param condition
float See Table 'Setting parameter as function of the marker states'. Range: 0 to 65535

set_variable(*variable: str*) → None

```
# SCPI: SEQuence:ITEM:MARKer:CONDition:VARiable
driver.sequence.item.marker.condition.set_variable(variable = 'abc')
```

Defines the value that is compared with the fixed values set with the command method RsPulseSeq.Sequence.Item.Marker. Condition.stringValue.

param variable
string

5.28.1.8 Ovl

SCPI Commands :

```
SEQuence:ITEM:OVL:VARiable
SEQuence:ITEM:OVL:WTIMe
```

class OvlCls

Ovl commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_variable() → str

```
# SCPI: SEQuence:ITEM:OVL:VARiable
value: str = driver.sequence.item.ovl.get_variable()
```

Sets a variable.

return
variable: string

get_wtime() → float

```
# SCPI: SEQuence:ITEM:OVL:WTIMe
value: float = driver.sequence.item.ovl.get_wtime()
```

Sets the duration of the overlay.

return
wtime: float Range: 0 to 3600, Unit: sec

set_variable(*variable: str*) → None

```
# SCPI: SEQuence:ITEM:OVL:VARiable
driver.sequence.item.ovl.set_variable(variable = 'abc')
```

Sets a variable.

param variable
string

set_wtime(*wtime: float*) → None

```
# SCPI: SEQuence:ITEM:OVL:WTIMe
driver.sequence.item.ovl.set_wtime(wtime = 1.0)
```

Sets the duration of the overlay.

param wtime
float Range: 0 to 3600, Unit: sec

5.28.1.9 Phase

SCPI Command :

```
SEQuence:ITEM:PHASe:OFFSet
```

class PhaseCls

Phase commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_offset() → float

```
# SCPI: SEQuence:ITEM:PHASe:OFFSet
value: float = driver.sequence.item.phase.get_offset()
```

Sets a phase offset.

return
offset: float Range: -180 to 180

set_offset(*offset: float*) → None

```
# SCPI: SEQuence:ITEM:PHASe:OFFSet
driver.sequence.item.phase.set_offset(offset = 1.0)
```

Sets a phase offset.

param offset
float Range: -180 to 180

5.28.1.10 Rep

SCPI Commands :

```
SEQuence:ITEM:REP:TYPE
SEQuence:ITEM:REP:VARiable
```

class RepCls

Rep commands group definition. 7 total commands, 1 Subgroups, 2 group commands

get_type_py() → RepetitionType

```
# SCPI: SEQuence:ITEM:REP:TYPE
value: enums.RepetitionType = driver.sequence.item.rep.get_type_py()
```

Sets how the repetition number is defined.

return
type_py: FIXed| VARiable| DURation

get_variable() → str

```
# SCPI: SEQuence:ITEM:REP:VARiable
value: str = driver.sequence.item.rep.get_variable()
```

Seta a repetition variable.

return
variable: string

set_type_py(*type_py: RepetitionType*) → None

```
# SCPI: SEQuence:ITEM:REP:TYPE
driver.sequence.item.rep.set_type_py(type_py = enums.RepetitionType.DURation)
```

Sets how the repetition number is defined.

param type_py
FIXed| VARiable| DURation

set_variable(*variable: str*) → None

```
# SCPI: SEQuence:ITEM:REP:VARiable
driver.sequence.item.rep.set_variable(variable = 'abc')
```

Seta a repetition variable.

param variable
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sequence.item.rep.clone()
```

Subgroups

5.28.1.10.1 Count

SCPI Commands :

```
SEQuence:ITEM:REP:COUNt:DURation
SEQuence:ITEM:REP:COUNt:FIXed
SEQuence:ITEM:REP:COUNt:MAXimum
SEQuence:ITEM:REP:COUNt:MINimum
SEQuence:ITEM:REP:COUNt:STEP
```

class CountCls

Count commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_duration() → float

```
# SCPI: SEQuence:ITEM:REP:COUNt:DURation
value: float = driver.sequence.item.rep.count.get_duration()
```

Sets a time duration.

return
duration: float Range: 0 to 1e+09, Unit: sec

get_fixed() → float

```
# SCPI: SEQuence:ITEM:REP:COUNt:FIXed
value: float = driver.sequence.item.rep.count.get_fixed()
```

Sets the repetition number as a numeric value.

return
fixed: float Range: 1 to 65535

get_maximum() → float

```
# SCPI: SEQuence:ITEM:REP:COUNt:MAXimum
value: float = driver.sequence.item.rep.count.get_maximum()
```

Sets the value range of the repetition count.

return
maximum: float Range: 1 to 65535

get_minimum() → float

```
# SCPI: SEQuence:ITEM:REP:COUNt:MINimum
value: float = driver.sequence.item.rep.count.get_minimum()
```

Sets the value range of the repetition count.

return
minimum: No help available

get_step() → float

```
# SCPI: SEQuence:ITEM:REP:COUNt:STEP
value: float = driver.sequence.item.rep.count.get_step()
```

Sets the repetition count granularity.

return
step: float Range: 1 to 65535

set_duration(*duration: float*) → None

```
# SCPI: SEQuence:ITEM:REP:COUNt:DURation
driver.sequence.item.rep.count.set_duration(duration = 1.0)
```

Sets a time duration.

param duration
float Range: 0 to 1e+09, Unit: sec

set_fixed(*fixed: float*) → None

```
# SCPI: SEQuence:ITEM:REP:COUNt:FIXed
driver.sequence.item.rep.count.set_fixed(fixed = 1.0)
```

Sets the repetition number as a numeric value.

param fixed
float Range: 1 to 65535

set_maximum(*maximum: float*) → None

```
# SCPI: SEQuence:ITEM:REP:COUNt:MAXimum
driver.sequence.item.rep.count.set_maximum(maximum = 1.0)
```

Sets the value range of the repetition count.

param maximum
float Range: 1 to 65535

set_minimum(*minimum: float*) → None

```
# SCPI: SEQuence:ITEM:REP:COUNt:MINimum
driver.sequence.item.rep.count.set_minimum(minimum = 1.0)
```

Sets the value range of the repetition count.

param minimum
float Range: 1 to 65535

set_step(*step: float*) → None

```
# SCPI: SEQuence:ITEM:REP:COUNt:STEP
driver.sequence.item.rep.count.set_step(step = 1.0)
```

Sets the repetition count granularity.

param step
float Range: 1 to 65535

5.28.2 Phase

SCPI Command :

```
SEQuence:PHASe:MODE
```

class PhaseCls

Phase commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → PhaseMode

```
# SCPI: SEQuence:PHASe:MODE
value: enums.PhaseMode = driver.sequence.phase.get_mode()
```

Defines how the phase is set at each pulse start.

return
mode: ABSolute| CONTinuous| MEMory

set_mode(*mode: PhaseMode*) → None

```
# SCPI: SEQuence:PHASe:MODE
driver.sequence.phase.set_mode(mode = enums.PhaseMode.ABSolute)
```

Defines how the phase is set at each pulse start.

param mode
ABSolute| CONTinuous| MEMory

5.28.3 Time

SCPI Command :

```
SEQuence:TIME:MODE
```

class TimeCls

Time commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → TimeMode

```
# SCPI: SEQuence:TIME:MODE
value: enums.TimeMode = driver.sequence.time.get_mode()
```

Switches between time-based (PRI) and frequency-based (PRF) pulse repetition definition.

return
mode: PRI| PRF

set_mode(*mode: TimeMode*) → None

```
# SCPI: SEQuence:TIME:MODE
driver.sequence.time.set_mode(mode = enums.TimeMode.PRF)
```

Switches between time-based (PRI) and frequency-based (PRF) pulse repetition definition.

param mode
PRI| PRF

5.29 Setup

SCPI Commands :

```
SETup:ADD
SETup:DELete
SETup:EXPort
SETup:IMPort
SETup:BBSYnc
SETup:COUNt
SETup:LIST
SETup:SELect
```

class SetupCls

Setup commands group definition. 16 total commands, 4 Subgroups, 8 group commands

delete(*delete: float*) → None

```
# SCPI: SETup:DELete
driver.setup.delete(delete = 1.0)
```

No command help available

param delete
No help available

export(*export: str*) → None

```
# SCPI: SETup:EXPort
driver.setup.export(export = 'abc')
```

No command help available

param export
No help available

get_bb_sync() → BbSync

```
# SCPI: SETup:BBSYnc
value: enums.BbSync = driver.setup.get_bb_sync()
```

Sets if and which method the signal generator uses to synchronize the signals in the baseband domain. Relevant in multi-instrument setups where the signals of the different emitters are generated in different paths and different signal generators.

return
bb_sync: UNSYnc| TRIGger| CTRigger UNSYnc Unsynchronized baseband generators TRIGger Synchronized setup, where the instruments are connected in a primary/secondary chain. General-purpose trigger signal is used. CTRigger Synchronized primary/secondary setup, that uses a dedicated common trigger signal.

get_count() → float

```
# SCPI: SETup:COUNt
value: float = driver.setup.get_count()
```

Queries the number of existing items.

return
count: integer

get_list_py() → List[str]

```
# SCPI: SETup:LIST
value: List[str] = driver.setup.get_list_py()
```

Queries the name of the available hardware setups.

return
list_py: 'Setup#1','Setup#2',...

get_select() → str

```
# SCPI: SETup:SELect
value: str = driver.setup.get_select()
```

Selects the item to which the subsequent commands apply.

return
select: float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

set_add(*add: str*) → None

```
# SCPI: SETup:ADD
driver.setup.set_add(add = 'abc')
```

No command help available

param add
No help available

set_bb_sync(*bb_sync: BbSync*) → None

```
# SCPI: SETup:BBSYnc
driver.setup.set_bb_sync(bb_sync = enums.BbSync.CTRigger)
```

Sets if and which method the signal generator uses to synchronize the signals in the baseband domain. Relevant in multi-instrument setups where the signals of the different emitters are generated in different paths and different signal generators.

param bb_sync
UNSYnc| TRIGger| CTRigger UNSYnc Unsynchronized baseband generators TRIGger Synchronized setup, where the instruments are connected in a primary/secondary chain. General-purpose trigger signal is used. CTRigger Synchronized primary/secondary setup, that uses a dedicated common trigger signal.

set_import_py(*import_py: str*) → None

```
# SCPI: SETup:IMPort
driver.setup.set_import_py(import_py = 'abc')
```

No command help available

param import_py
No help available

set_select(*select: str*) → None

```
# SCPI: SETup:SELect
driver.setup.set_select(select = 'abc')
```

Selects the item to which the subsequent commands apply.

param select
float Item number within the range 1 to ...:COUNt. For example, method RsPulseSeq.Sequence.Item.count. Range: 1 to 4096

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.setup.clone()
```

Subgroups

5.29.1 HigHq

SCPI Commands :

```
SETup:HIGHq:ENABle
SETup:HIGHq:MODE
```

class HigHqCls

HigHq commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SETup:HIGHq:ENABle
value: bool = driver.setup.higHq.get_enable()
```

Sets the I/Q modulator of signal generator to work in a high quality mode.

return
enable: ON| OFF| 1| 0

get_mode() → HqMode

```
# SCPI: SETup:HIGHq:MODE
value: enums.HqMode = driver.setup.higHq.get_mode()
```

Defines which high-quality mode is used. Requires that method RsPulseSeq.Setup.HigHq.enable is set to ON (see method RsPulseSeq.Setup.HigHq.enable) .

return
mode: NORMal| TABLe NORM Enables compensation for I/Q skew and frequency response correction. This mode generates a flat signal over a large bandwidth but requires longer setting time and can lead to signal interruption. TABL This mode provides optimization while maintaining settling time.

set_enable(*enable: bool*) → None

```
# SCPI: SETup:HIGHq:ENABle
driver.setup.higHq.set_enable(enable = False)
```

Sets the I/Q modulator of signal generator to work in a high quality mode.

param enable
ON| OFF| 1| 0

set_mode(*mode: HqMode*) → None

```
# SCPI: SETup:HIGHq:MODE
driver.setup.higHq.set_mode(mode = enums.HqMode.NORMal)
```

Defines which high-quality mode is used. Requires that method RsPulseSeq.Setup.HigHq.enable is set to ON (see method RsPulseSeq.Setup.HigHq.enable) .

param mode
NORMal| TABLe NORM Enables compensation for I/Q skew and frequency response correction. This mode generates a flat signal over a large bandwidth but requires longer setting time and can lead to signal interruption. TABL This mode provides optimization while maintaining settling time.

5.29.2 Locpl

SCPI Command :

```
SETup:LOCPl:ENABle
```

class LocplCls

Locpl commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SETup:LOCPl:ENABle
value: bool = driver.setup.locpl.get_enable()
```

Couples the LO signals.

return
enable: ON| OFF| 1| 0

set_enable(*enable: bool*) → None

```
# SCPI: SETup:LOCPl:ENABle
driver.setup.locpl.set_enable(enable = False)
```

Couples the LO signals.

param enable
ON| OFF| 1| 0

5.29.3 Pmod

SCPI Commands :

```
SETup:PMOD:DIRection
SETup:PMOD:ENABle
```

class PmodCls

Pmod commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_direction() → SourceInt

```
# SCPI: SETup:PMOD:DIRection
value: enums.SourceInt = driver.setup.pmod.get_direction()
```

No command help available

return
direction: No help available

get_enable() → bool

```
# SCPI: SETup:PMOD:ENABle
value: bool = driver.setup.pmod.get_enable()
```

Enables the activation of the pulse modulator for improving the on/off ratio.

return
enable: ON| OFF| 1| 0

set_direction(*direction: SourceInt*) → None

```
# SCPI: SETup:PMOD:DIRection
driver.setup.pmod.set_direction(direction = enums.SourceInt.EXTernal)
```

No command help available

param direction
No help available

set_enable(*enable: bool*) → None

```
# SCPI: SETup:PMOD:ENABle
driver.setup.pmod.set_enable(enable = False)
```

Enables the activation of the pulse modulator for improving the on/off ratio.

param enable
ON| OFF| 1| 0

5.29.4 RfAlign

SCPI Commands :

```
SETup:RFALign:INSTrument
SETup:RFALign:SETup
```

class RfAlignCls

RfAlign commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_instrument() → str

```
# SCPI: SETup:RFALign:INSTrument
value: str = driver.setup.rfAlign.get_instrument()
```

No command help available

return
instrument: No help available

get_setup() → str

```
# SCPI: SETup:RFALign:SETup
value: str = driver.setup.rfAlign.get_setup()
```

No command help available

return
setup: No help available

set_instrument(*instrument: str*) → None

```
# SCPI: SETup:RFALign:INSTrument
driver.setup.rfAlign.set_instrument(instrument = 'abc')
```

No command help available

param instrument
No help available

set_setup(*setup: str*) → None

```
# SCPI: SETup:RFALign:SETup
driver.setup.rfAlign.set_setup(setup = 'abc')
```

No command help available

param setup
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.setup.rfAlign.clone()
```

Subgroups

5.29.4.1 ImportPy

SCPI Command :

```
SETup:RFALign:IMPort
```

class ImportPyCls

ImportPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SETup:RFALign:IMPort
driver.setup.rfAlign.importPy.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SETup:RFALign:IMPort
driver.setup.rfAlign.importPy.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.30 Status

class StatusCls

Status commands group definition. 10 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.clone()
```

Subgroups

5.30.1 Operation

SCPI Commands :

```
STATus:OPERation:CONDition
STATus:OPERation:ENABle
STATus:OPERation:EVENt
STATus:OPERation:NTRansition
STATus:OPERation:PTRansition
```

class OperationCls

Operation commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_condition() → float

```
# SCPI: STATus:OPERation:CONDition
value: float = driver.status.operation.get_condition()
```

Queries the content of the CONDition part of the STATus:OPERation register. This part contains information on the action currently being performed in the instrument. The content is not deleted after being read out because it indicates the current status.

return
condition: float

get_enable() → bool

```
# SCPI: STATus:OPERation:ENABle
value: bool = driver.status.operation.get_enable()
```

No command help available

return
enable: No help available

get_event() → float

```
# SCPI: STATus:OPERation:EVENt
value: float = driver.status.operation.get_event()
```

No command help available

return
event: No help available

get_ntransition() → float

```
# SCPI: STATus:OPERation:NTRansition
value: float = driver.status.operation.get_ntransition()
```

No command help available

return
ntransition: No help available

get_ptransition() → float

```
# SCPI: STATus:OPERation:PTRansition
value: float = driver.status.operation.get_ptransition()
```

No command help available

return
ptransition: No help available

set_enable(*enable: bool*) → None

```
# SCPI: STATus:OPERation:ENABle
driver.status.operation.set_enable(enable = False)
```

No command help available

param enable
No help available

set_ntransition(*ntransition: float*) → None

```
# SCPI: STATus:OPERation:NTRansition
driver.status.operation.set_ntransition(ntransition = 1.0)
```

No command help available

param ntransition
No help available

set_ptransition(*ptransition: float*) → None

```
# SCPI: STATus:OPERation:PTRansition
driver.status.operation.set_ptransition(ptransition = 1.0)
```

No command help available

param ptransition
No help available

5.30.2 Quesation

SCPI Commands :

```
STATus:QUESation:CONDition
STATus:QUESation:ENABle
STATus:QUESation:EVENt
STATus:QUESation:NTRansition
STATus:QUESation:PTRansition
```

class QuesationCls

Quesation commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_condition() → float

```
# SCPI: STATus:QUESation:CONDition
value: float = driver.status.quesation.get_condition()
```

No command help available

return
condition: No help available

get_enable() → bool

```
# SCPI: STATus:QUESation:ENABle
value: bool = driver.status.quesation.get_enable()
```

No command help available

return
enable: No help available

get_event() → float

```
# SCPI: STATus:QUESation:EVENt
value: float = driver.status.quesation.get_event()
```

No command help available

return
event: No help available

get_ntransition() → float

```
# SCPI: STATus:QUESation:NTRansition
value: float = driver.status.quesation.get_ntransition()
```

No command help available

return
ntransition: No help available

get_ptransition() → float

```
# SCPI: STATus:QUESation:PTRansition
value: float = driver.status.quesation.get_ptransition()
```

No command help available

return
ptransition: No help available

set_enable(*enable: bool*) → None

```
# SCPI: STATus:QUESation:ENABle
driver.status.quesation.set_enable(enable = False)
```

No command help available

param enable
No help available

set_ntransition(*ntransition: float*) → None

```
# SCPI: STATus:QUESation:NTRansition
driver.status.quesation.set_ntransition(ntransition = 1.0)
```

No command help available

param ntransition
No help available

set_ptransition(*ptransition: float*) → None

```
# SCPI: STATus:QUESation:PTRansition
driver.status.quesation.set_ptransition(ptransition = 1.0)
```

No command help available

param ptransition
No help available

5.31 System

SCPI Command :

```
SYSTem:PROGress
```

class SystemCls

System commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_progress() → float

```
# SCPI: SYSTem:PROGress
value: float = driver.system.get_progress()
```

Queries the signal generation progress status.

return
progress: float

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.clone()
```

Subgroups

5.31.1 Error

SCPI Commands :

```
SYSTem:ERRor:ALL
SYSTem:ERRor:COUNt
```

class ErrorCls

Error commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_all() → str

```
# SCPI: SYSTem:ERRor:ALL
value: str = driver.system.error.get_all()
```

Queries the error/event queue for all unread items and removes them from the queue. The response is a comma-separated list of error number and a short description of the error in FIFO order. Positive error numbers are instrument-dependent. Negative error numbers are reserved by the SCPI standard.

return
all_py: string List of: Error/event_number,'Error/event_description[;Device-dependent info]' If the queue is empty, the response is 0,'No error'

get_count() → str

```
# SCPI: SYSTem:ERRor:COUNt
value: str = driver.system.error.get_count()
```

Queries the number of entries in the error queue. If the error queue is empty, '0' is returned.

return
count: string

5.32 Waveform

SCPI Commands :

```
WAVeform:CATalog
WAVeform:COMMent
WAVeform:CREate
WAVeform:NAME
WAVeform:REMove
WAVeform:SELect
WAVeform:SIGCont
WAVeform:TYPE
```

class WaveformCls

Waveform commands group definition. 30 total commands, 8 Subgroups, 8 group commands

get_catalog() → str

```
# SCPI: WAVeform:CATalog
value: str = driver.waveform.get_catalog()
```

Queries the available repository elements in the database.

return
catalog: string

get_comment() → str

```
# SCPI: WAVeform:COMMent
value: str = driver.waveform.get_comment()
```

Adds a description to the selected repository element.

return
comment: string

get_name() → str

```
# SCPI: WAVeform:NAME
value: str = driver.waveform.get_name()
```

Renames the selected repository element.

return
name: string Must be unique for the particular type of repository elements. May contain empty spaces.

get_select() → str

```
# SCPI: WAVeform:SELect
value: str = driver.waveform.get_select()
```

Selects the repository element to which the subsequent commands apply.

return
select: string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

get_sig_cont() → SigCont

```
# SCPI: WAVeform:SIGCont
value: enums.SigCont = driver.waveform.get_sig_cont()
```

Defines the waveform signal type.

return
sig_cont: PULSe| COMM

get_type_py() → WaveformType

```
# SCPI: WAVeform:TYPE
value: enums.WaveformType = driver.waveform.get_type_py()
```

Sets the type of the waveform.

return
type_py: CW| NOISe| WAVeform| USER| BEMitter| MT| PDW| AIF| APDW| IQDW

set_comment(*comment: str*) → None

```
# SCPI: WAVeform:COMMent
driver.waveform.set_comment(comment = 'abc')
```

Adds a description to the selected repository element.

param comment
string

set_create(*create: str*) → None

```
# SCPI: WAVeform:CREate
driver.waveform.set_create(create = 'abc')
```

Creates a repository element with the selected name.

param create
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_name(*name: str*) → None

```
# SCPI: WAVeform:NAME
driver.waveform.set_name(name = 'abc')
```

Renames the selected repository element.

param name
string Must be unique for the particular type of repository elements. May contain empty spaces.

set_remove(*remove: str*) → None

```
# SCPI: WAVeform:REMove
driver.waveform.set_remove(remove = 'abc')
```

Removes the selected element from the workspace. The element must not reference any child elements. Remove the referenced elements first.

param remove
No help available

set_select(*select: str*) → None

```
# SCPI: WAVeform:SELect
driver.waveform.set_select(select = 'abc')
```

Selects the repository element to which the subsequent commands apply.

param select
string Element name, as defined with the ...:CREate or ...:NAME command. To query the existing elements, use the ...:CATalog? command. For example, method RsPulseSeq.Repository.catalog.

set_sig_cont(*sig_cont: SigCont*) → None

```
# SCPI: WAVeform:SIGCont
driver.waveform.set_sig_cont(sig_cont = enums.SigCont.COMM)
```

Defines the waveform signal type.

param sig_cont
PULSe| COMM

set_type_py(*type_py: WaveformType*) → None

```
# SCPI: WAVeform:TYPE
driver.waveform.set_type_py(type_py = enums.WaveformType.AIF)
```

Sets the type of the waveform.

param type_py
CW| NOISe| WAVeform| USER| BEMitter| MT| PDW| AIF| APDW| IQDW

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.waveform.clone()
```

Subgroups

5.32.1 Bemitter

SCPI Commands :

```
WAVeform:BEMitter:BWIDth
WAVeform:BEMitter:COUNt
WAVeform:BEMitter:DURation
```

class BemitterCls

Bemitter commands group definition. 8 total commands, 3 Subgroups, 3 group commands

get_bandwidth() → float

```
# SCPI: WAVeform:BEMitter:BWIDth
value: float = driver.waveform.bemitter.get_bandwidth()
```

Sets the maximum permissible bandwidth of the resulting signal.

return
bwidth: float Range: 1000 to 2e+09

get_count() → float

```
# SCPI: WAVeform:BEMitter:COUNt
value: float = driver.waveform.bemitter.get_count()
```

Sets the number of background emitters.

return
count: integer Range: 1 to 255

get_duration() → float

```
# SCPI: WAVeform:BEMitter:DURation
value: float = driver.waveform.bemitter.get_duration()
```

Sets the signal duration.

return
duration: float Range: 0.0001 to 1

set_bandwidth(*bwidth: float*) → None

```
# SCPI: WAVeform:BEMitter:BWIDth
driver.waveform.bemitter.set_bandwidth(bwidth = 1.0)
```

Sets the maximum permissible bandwidth of the resulting signal.

param bwidth
float Range: 1000 to 2e+09

set_count(*count: float*) → None

```
# SCPI: WAVeform:BEMitter:COUNt
driver.waveform.bemitter.set_count(count = 1.0)
```

Sets the number of background emitters.

param count
integer Range: 1 to 255

set_duration(*duration: float*) → None

```
# SCPI: WAVeform:BEMitter:DURation
driver.waveform.bemitter.set_duration(duration = 1.0)
```

Sets the signal duration.

param duration
float Range: 0.0001 to 1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.waveform.bemitter.clone()
```

Subgroups

5.32.1.1 Level

SCPI Command :

```
WAVeform:BEMitter:LEVel:RANGe
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_range() → float

```
# SCPI: WAVeform:BEMitter:LEVel:RANGe
value: float = driver.waveform.bemitter.level.get_range()
```

Sets the maximum level difference between the strongest and the weakest emitter.

return
range_py: float Range: 0 to 90

set_range(*range_py: float*) → None

```
# SCPI: WAVeform:BEMitter:LEVel:RANGe
driver.waveform.bemitter.level.set_range(range_py = 1.0)
```

Sets the maximum level difference between the strongest and the weakest emitter.

param range_py
float Range: 0 to 90

5.32.1.2 Pri

class PriCls

Pri commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.waveform.bemitter.pri.clone()
```

Subgroups

5.32.1.2.1 Ratio

SCPI Commands :

```
WAVeform:BEMitter:PRI:RATio:MAXimum
WAVeform:BEMitter:PRI:RATio:MINimum
```

class RatioCls

Ratio commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_maximum() → float

```
# SCPI: WAVeform:BEMitter:PRI:RATio:MAXimum
value: float = driver.waveform.bemitter.pri.ratio.get_maximum()
```

Sets the value range for the PRI/PW ratio.

return
maximum: float Range: 10 to 1000

get_minimum() → float

```
# SCPI: WAVeform:BEMitter:PRI:RATio:MINimum
value: float = driver.waveform.bemitter.pri.ratio.get_minimum()
```

Sets the value range for the PRI/PW ratio.

return
minimum: No help available

set_maximum(*maximum: float*) → None

```
# SCPI: WAVeform:BEMitter:PRI:RATio:MAXimum
driver.waveform.bemitter.pri.ratio.set_maximum(maximum = 1.0)
```

Sets the value range for the PRI/PW ratio.

param maximum
float Range: 10 to 1000

set_minimum(*minimum: float*) → None

```
# SCPI: WAVeform:BEMitter:PRI:RATio:MINimum
driver.waveform.bemitter.pri.ratio.set_minimum(minimum = 1.0)
```

Sets the value range for the PRI/PW ratio.

param minimum
float Range: 10 to 1000

5.32.1.3 Pw

SCPI Commands :

```
WAVeform:BEMitter:PW:MAXimum
WAVeform:BEMitter:PW:MINimum
```

class PwCls

Pw commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_maximum() → float

```
# SCPI: WAVeform:BEMitter:PW:MAXimum
value: float = driver.waveform.bemitter.pw.get_maximum()
```

Sets the value range for the pulse width values.

return
maximum: float Range: 5e-05 to 1

get_minimum() → float

```
# SCPI: WAVeform:BEMitter:PW:MINimum
value: float = driver.waveform.bemitter.pw.get_minimum()
```

Sets the value range for the pulse width values.

return
minimum: No help available

set_maximum(*maximum: float*) → None

```
# SCPI: WAVeform:BEMitter:PW:MAXimum
driver.waveform.bemitter.pw.set_maximum(maximum = 1.0)
```

Sets the value range for the pulse width values.

param maximum
float Range: 5e-05 to 1

set_minimum(*minimum: float*) → None

```
# SCPI: WAVeform:BEMitter:PW:MINimum
driver.waveform.bemitter.pw.set_minimum(minimum = 1.0)
```

Sets the value range for the pulse width values.

param minimum
float Range: 5e-05 to 1

5.32.2 Iq

SCPI Command :

```
WAVeform:IQ:CLEar
```

class IqCls

Iq commands group definition. 1 total commands, 0 Subgroups, 1 group commands

clear() → None

```
# SCPI: WAVeform:IQ:CLEar
driver.waveform.iq.clear()
```

Removes the imported waveform or file with I/Q data.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: WAVeform:IQ:CLEar
driver.waveform.iq.clear_with_opc()
```

Removes the imported waveform or file with I/Q data.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.32.3 Level

SCPI Command :

```
WAVeform:LEVel:REFerence
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_reference() → float

```
# SCPI: WAVeform:LEVel:REFerence
value: float = driver.waveform.level.get_reference()
```

Queries the reference level.

return
reference: float Range: 0 to 100

set_reference(*reference: float*) → None

```
# SCPI: WAVeform:LEVel:REFerence
driver.waveform.level.set_reference(reference = 1.0)
```

Queries the reference level.

param reference
float Range: 0 to 100

5.32.4 Mt

SCPI Commands :

```
WAVeform:MT:COUNt
WAVeform:MT:SPACing
```

class MtCls

Mt commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_count() → float

```
# SCPI: WAVeform:MT:COUNt
value: float = driver.waveform.mt.get_count()
```

Sets the number of tones.

return
count: integer Range: 2 to 1024

get_spacing() → float

```
# SCPI: WAVeform:MT:SPACing
value: float = driver.waveform.mt.get_spacing()
```

Sets the tone spacing.

return
spacing: float Range: 100 to 1e+07

set_count(*count: float*) → None

```
# SCPI: WAVeform:MT:COUNt
driver.waveform.mt.set_count(count = 1.0)
```

Sets the number of tones.

param count
integer Range: 2 to 1024

set_spacing(*spacing: float*) → None

```
# SCPI: WAVeform:MT:SPACing
driver.waveform.mt.set_spacing(spacing = 1.0)
```

Sets the tone spacing.

param spacing
float Range: 100 to 1e+07

5.32.5 Noise

SCPI Command :

```
WAVeform:NOISe:BWIDth
```

class NoiseCls

Noise commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_bandwidth() → float

```
# SCPI: WAVeform:NOISe:BWIDth
value: float = driver.waveform.noise.get_bandwidth()
```

Sets the bandwidth of the generated AWGN waveform.

return
bwidth: float Range: 0 to 2e+09, Unit: Hz

set_bandwidth(*bwidth: float*) → None

```
# SCPI: WAVeform:NOISe:BWIDth
driver.waveform.noise.set_bandwidth(bwidth = 1.0)
```

Sets the bandwidth of the generated AWGN waveform.

param bwidth
float Range: 0 to 2e+09, Unit: Hz

5.32.6 Pdw

SCPI Command :

```
WAVeform:PDW:CENTer
```

class PdwCls

Pdw commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_center() → float

```
# SCPI: WAVeform:PDW:CENTer
value: float = driver.waveform.pdw.get_center()
```

During the import of PDW list files, the software evaluates the frequency information in the file and calculates the center frequency of all pulses. The center frequency is calculated as the middle frequency between the min and the max frequency values included in the PDW file. Chirp frequency deviations are also considered. The pulses are calculated relatively to this center frequency. If the actual frequency of the generator differs from the calculated one, use this command to set the center frequency of the generator.

return
center: float Range: 0 to 1e+11

set_center(*center: float*) → None

```
# SCPI: WAVeform:PDW:CENTer
driver.waveform.pdw.set_center(center = 1.0)
```

During the import of PDW list files, the software evaluates the frequency information in the file and calculates the center frequency of all pulses. The center frequency is calculated as the middle frequency between the min and the max frequency values included in the PDW file. Chirp frequency deviations are also considered. The pulses are calculated relatively to this center frequency. If the actual frequency of the generator differs from the calculated one, use this command to set the center frequency of the generator.

param center
float Range: 0 to 1e+11

5.32.7 View

SCPI Commands :

```
WAVeform:VIEW:XMODe
WAVeform:VIEW:YMODe
```

class ViewCls

View commands group definition. 6 total commands, 3 Subgroups, 2 group commands

set_xmode(*xmode: ViewXode*) → None

```
# SCPI: WAVeform:VIEW:XMODe
driver.waveform.view.set_xmode(xmode = enums.ViewXode.SAMPles)
```

Sets the units (time or samples) used on the x axis.

param xmode
SAMPles| TIME

set_ymode(*ymode: Ymode*) → None

```
# SCPI: WAVeform:VIEW:YMODe
driver.waveform.view.set_ymode(ymode = enums.Ymode.FREQuency)
```

Sets the view mode.

param ymode
IQ| MAGDb| MAGW| MAGV| PHASe| FREQuency| PAV

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.waveform.view.clone()
```

Subgroups

5.32.7.1 Get

SCPI Command :

```
WAVeform:VIEW:GET:DURation
```

class GetCls

Get commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_duration() → float

```
# SCPI: WAVeform:VIEW:GET:DURation
value: float = driver.waveform.view.get.get_duration()
```

No command help available

return
duration: No help available

5.32.7.2 Open

class OpenCls

Open commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.waveform.view.open.clone()
```

Subgroups

5.32.7.2.1 Window

SCPI Command :

```
WAVeform:VIEW:OPEN:WINDow
```

class WindowCls

Window commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: WAVeform:VIEW:OPEN:WINDow
driver.waveform.view.open.window.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: WAVeform:VIEW:OPEN:WINDow
driver.waveform.view.open.window.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

5.32.7.3 Zoom

SCPI Commands :

```
WAVeform:VIEW:ZOOM:POINt
WAVeform:VIEW:ZOOM:RANGe
```

class ZoomCls

Zoom commands group definition. 2 total commands, 0 Subgroups, 2 group commands

set_point(*point: float*) → None

```
# SCPI: WAVeform:VIEW:ZOOM:POINt
driver.waveform.view.zoom.set_point(point = 1.0)
```

Sets center point of the displayed area.

param point
float Always related to time Unit: s

set_range(*range_py: float*) → None

```
# SCPI: WAVeform:VIEW:ZOOM:RANGe
driver.waveform.view.zoom.set_range(range_py = 1.0)
```

Sets the displayed waveform part as a range around the selected center point, set with the command method RsPulseSeq. Scenario.Volatile.View.Zoom.point.

param range_py
float Expressed as a time span (units can be omitted) or as number of samples

5.32.8 Waveform

SCPI Commands :

```
WAVeform:WAVeform:CLEar
WAVeform:WAVeform:LOAD
```

class WaveformCls

Waveform commands group definition. 2 total commands, 0 Subgroups, 2 group commands

clear() → None

```
# SCPI: WAVeform:WAVeform:CLEar
driver.waveform.waveform.clear()
```

Removes the imported waveform or file with I/Q data.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: WAVeform:WAVeform:CLEar
driver.waveform.waveform.clear_with_opc()
```

Removes the imported waveform or file with I/Q data.

Same as clear, but waits for the operation to complete before continuing further. Use the RsPulseSeq.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

load(*load: str*) → None

```
# SCPI: WAVeform:WAVeform:LOAD
driver.waveform.waveform.load(load = 'abc')
```

Load the selected waveform file (*.wv) , see Table 'Supported file types'.

param load
string Complete file path with file name and extension

CHAPTER

SIX

RSPULSESEQ UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsPulseSeq.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsPulseSeq.utilities.logger`

property driver_version: str

Returns the instrument driver version.

property idn_string: str

Returns instrument's identification string - the response on the SCPI command *IDN?

property manufacturer: str

Returns manufacturer of the instrument.

property full_instrument_model_name: str

Returns the current instrument's full name e.g. 'FSW26'.

property instrument_model_name: str

Returns the current instrument's family name e.g. 'FSW'.

property supported_models: List[str]

Returns a list of the instrument models supported by this instrument driver.

property instrument_firmware_version: str

Returns instrument's firmware version.

property instrument_serial_number: str

Returns instrument's serial_number.

query_opc(*timeout: int = 0*) → int

SCPI command: *OPC? Queries the instrument's OPC bit and hence it waits until the instrument reports operation complete. If you define timeout > 0, the VISA timeout is set to that value just for this method call.

property instrument_status_checking: bool

Sets / returns Instrument Status Checking. When True (default is True), all the driver methods and properties are sending "SYSTem:ERRor?" at the end to immediately react on error that might have occurred. We recommend to keep the state checking ON all the time. Switch it OFF only in rare cases when you require maximum speed. The default state after initializing the session is ON.

property encoding: str

Returns string<=>bytes encoding of the session.

property opc_query_after_write: bool

Sets / returns Instrument *OPC? query sending after each command write. When True, (default is False) the driver sends *OPC? every time a write command is performed. Use this if you want to make sure your sequence is performed command-after-command.

property bin_float_numbers_format: BinFloatFormat

Sets / returns format of float numbers when transferred as binary data.

property bin_int_numbers_format: BinIntFormat

Sets / returns format of integer numbers when transferred as binary data.

clear_status() → None

Clears instrument's status system, the session's I/O buffers and the instrument's error queue.

query_all_errors() → List[str]

Queries and clears all the errors from the instrument's error queue. The method returns list of strings as error messages. If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty. If you want to include the error codes, call the query_all_errors_with_codes()

query_all_errors_with_codes() → List[Tuple[int, str]]

Queries and clears all the errors from the instrument's error queue. The method returns list of tuples (code: int, message: str). If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty.

property instrument_options: List[str]

Returns all the instrument options. The options are sorted in the ascending order starting with K-options and continuing with B-options.

reset() → None

SCPI command: *RST Sends *RST command + calls the clear_status().

default_instrument_setup() → None

Custom steps performed at the init and at the reset().

self_test(*timeout: int = None*) → Tuple[int, str]

SCPI command: *TST? Performs instrument's self-test. Returns tuple (code:int, message: str). Code 0 means the self-test passed. You can define the custom timeout in milliseconds. If you do not define it, the default selftest timeout is used (usually 60 secs).

is_connection_active() → bool

Returns true, if the VISA connection is active and the communication with the instrument still works.

reconnect(*force_close: bool = False*) → bool

If the connection is not active, the method tries to reconnect to the device If the connection is active, and force_close is False, the method does nothing. If the connection is active, and force_close is True, the method closes, and opens the session again. Returns True, if the reconnection has been performed.

property resource_name: int

Returns the resource name used in the constructor

property opc_timeout: int

Sets / returns timeout in milliseconds for all the operations that use OPC synchronization.

property visa_timeout: int

Sets / returns visa IO timeout in milliseconds.

property data_chunk_size: int

Sets / returns the maximum size of one block transferred during write/read operations

property visa_manufacturer: int

Returns the manufacturer of the current VISA session.

process_all_commands() → None

SCPI command: *WAI Stops further commands processing until all commands sent before *WAI have been executed.

write_str(*cmd: str*) → None

Writes the command to the instrument.

write(*cmd: str*) → None

This method is an alias to the write_str(). Writes the command to the instrument as string.

write_int(*cmd: str*, *param: int*) → None

Writes the command to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2'

write_int_with_opc(*cmd: str*, *param: int*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2' If you do not provide timeout, the method uses current opc_timeout.

write_float(*cmd: str*, *param: float*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6'

write_float_with_opc(*cmd: str*, *param: float*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6' If you do not provide timeout, the method uses current opc_timeout.

write_bool(*cmd: str*, *param: bool*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON'

write_bool_with_opc(*cmd: str*, *param: bool*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON' If you do not provide timeout, the method uses current opc_timeout.

query_str(*query: str*) → str

Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query(*query: str*) → str

This method is an alias to the query_str(). Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query_bool(*query: str*) → bool

Sends the query to the instrument and returns the response as boolean.

query_int(*query: str*) → int

Sends the query to the instrument and returns the response as integer.

query_float(*query: str*) → float

Sends the query to the instrument and returns the response as float.

write_str_with_opc(*cmd: str*, *timeout: int = None*) → None

Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

write_with_opc(*cmd: str*, *timeout: int = None*) → None

This method is an alias to the write_str_with_opc(). Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

query_str_with_opc(*query: str*, *timeout: int = None*) → str

Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_with_opc(*query: str*, *timeout: int = None*) → str

This method is an alias to the query_str_with_opc(). Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_bool_with_opc(*query: str*, *timeout: int = None*) → bool

Sends the opc-synced query to the instrument and returns the response as boolean. If you do not provide timeout, the method uses current opc_timeout.

query_int_with_opc(*query: str*, *timeout: int = None*) → int

Sends the opc-synced query to the instrument and returns the response as integer. If you do not provide timeout, the method uses current opc_timeout.

query_float_with_opc(*query: str*, *timeout: int = None*) → float

Sends the opc-synced query to the instrument and returns the response as float. If you do not provide timeout, the method uses current opc_timeout.

write_bin_block(*cmd: str*, *payload: bytes*) → None

Writes all the payload as binary data block to the instrument. The binary data header is added at the beginning of the transmission automatically, do not include it in the payload!!!

query_bin_block(*query: str*) → bytes

Queries binary data block to bytes. Throws an exception if the returned data was not a binary data. Returns data:bytes

query_bin_block_with_opc(*query: str*, *timeout: int = None*) → bytes

Sends a OPC-synced query and returns binary data block to bytes. If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_float_list(*query: str*) → List[float]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_float_list_with_opc(*query: str*, *timeout: int = None*) → List[float]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_int_list(*query: str*) → List[int]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_int_list_with_opc(*query: str*, *timeout: int = None*) → List[int]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_block_to_file(*query: str*, *file_path: str*, *append: bool = False*) → None

Queries binary data block to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data. Example for transferring a file from Instrument -> PC: query = f'MMEM:DATA? '{INSTR_FILE_PATH}'". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

query_bin_block_to_file_with_opc(*query: str*, *file_path: str*, *append: bool = False*, *timeout: int = None*) → None

Sends a OPC-synced query and writes the returned data to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data.

write_bin_block_from_file(*cmd: str*, *file_path: str*) → None

Writes data from the file as binary data block to the instrument using the provided command. Example for transferring a file from PC -> Instrument: cmd = f'MMEM:DATA '{INSTR_FILE_PATH}',". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

send_file_from_pc_to_instrument(*source_pc_file: str*, *target_instr_file: str*) → None

SCPI Command: MMEM:DATA

Sends file from PC to the instrument

read_file_from_instrument_to_pc(*source_instr_file: str*, *target_pc_file: str*, *append_to_pc_file: bool = False*) → None

SCPI Command: MMEM:DATA?

Reads file from instrument to the PC.

Set the `append_to_pc_file` to True if you want to append the read content to the end of the existing PC file

get_last_sent_cmd() → str

Returns the last commands sent to the instrument. Only works in simulation mode

go_to_local() → None

Puts the instrument into local state.

go_to_remote() → None

Puts the instrument into remote state.

get_lock() → RLock

Returns the thread lock for the current session.

By default:

- If you create standard new RsPulseSeq instance with new VISA session, the session gets a new thread lock. You can assign it to other RsPulseSeq sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsPulseSeq from an existing session, the thread lock is shared automatically making both instances multi-thread safe.

You can always assign new thread lock by calling `driver.utilities.assign_lock()`

assign_lock(*lock: RLock*) → None

Assigns the provided thread lock.

clear_lock()

Clears the existing thread lock, making the current session thread-independent from others that might share the current thread lock.

sync_from(*source: Utilities*) → None

Synchronises these Utils with the source.

CHAPTER

SEVEN

RSPULSESEQ LOGGER

Check the usage in the Getting Started chapter *here*.

class ScpiLogger

Base class for SCPI logging

mode

Sets the logging ON or OFF. Additionally, you can set the logging ON only for errors. Possible values:

- LoggingMode.Off - logging is switched OFF
- LoggingMode.On - logging is switched ON
- LoggingMode.Errors - logging is switched ON, but only for error entries
- LoggingMode.Default - sets the logging to default - the value you have set with logger.default_mode

default_mode

Sets / returns the default logging mode. You can recall the default mode by calling the logger.mode = LoggingMode.Default.

Data Type
LoggingMode

device_name: str

Use this property to change the resource name in the log from the default Resource Name (e.g. TCPIP::192.168.2.101::INSTR) to another name e.g. 'MySigGen1'.

set_logging_target(*target*, *console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target - the target must implement write() and flush(). You can optionally set the console and UDP logging ON or OFF. This method switches the logging target global OFF.

get_logging_target()

Based on the global_mode, it returns the logging target: either the local or the global one.

set_logging_target_global(*console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target to global. The global target must be defined. You can optionally set the console and UDP logging ON or OFF.

log_to_console

Returns logging to console status.

log_to_udp

Returns logging to UDP status.

log_to_console_and_udp

Returns true, if both logging to UDP and console in are True.

info_raw(*log_entry: str*, *add_new_line: bool = True*) → None

Method for logging the raw string without any formatting.

info(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one info entry. For binary log_string, use the info_bin()

error(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one error entry.

set_relative_timestamp(*timestamp: datetime*) → None

If set, the further timestamps will be relative to the entered time.

set_relative_timestamp_now() → None

Sets the relative timestamp to the current time.

get_relative_timestamp() → datetime

Based on the global_mode, it returns the relative timestamp: either the local or the global one.

clear_relative_timestamp() → None

Clears the reference time, and the further logging continues with absolute times.

flush() → None

Flush all the entries.

log_status_check_ok

Sets / returns the current status of status checking OK. If True (default), the log contains logging of the status checking 'Status check: OK'. If False, the 'Status check: OK' is skipped - the log is more compact. Errors will still be logged.

clear_cached_entries() → None

Clears potential cached log entries. Cached log entries are generated when the Logging is ON, but no target has been defined yet.

set_format_string(*value: str*, *line_divider: str = '\n'*) → None

Sets new format string and line divider. If you just want to set the line divider, set the format string value=None The original format string is: `PAD_LEFT12(%START_TIME%) PAD_LEFT25(%DEVICE_NAME%) PAD_LEFT12(%DURATION%) %LOG_STRING_INFO%: %LOG_STRING%`

restore_format_string() → None

Restores the original format string and the line divider to LF

abbreviated_max_len_ascii: int

Defines the maximum length of one ASCII log entry. Default value is 200 characters.

abbreviated_max_len_bin: int

Defines the maximum length of one Binary log entry. Default value is 2048 bytes.

abbreviated_max_len_list: int

Defines the maximum length of one list entry. Default value is 100 elements.

bin_line_block_size: int

Defines number of bytes to display in one line. Default value is 16 bytes.

udp_port

Returns udp logging port.

target_auto_flushing

Returns status of the auto-flushing for the logging target.

CHAPTER

EIGHT

RSPULSESEQ EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

NINE

INDEX

INDEX

A

B

C

D

E

F

G

I

L

M

P

R

S

T

U

W