# RsSmab

***Release 5.10.121.23***

**Rohde & Schwarz**

**Mar 25, 2024**

# CONTENTS:


SMAB

CHAPTER

ONE

# REVISION HISTORY

## 1.1 RsSmab

Rohde & Schwarz SMA100B Microwave Signal Generator RsSmab instrument driver.

Basic Hello-World code:

```
from RsSmab import *

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

Supported instruments: SMA100B

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

Documentation: https://RsSmab.readthedocs.io/

Examples: https://github.com/Rohde-Schwarz/Examples/tree/main/SignalGenerators/Python/RsSmab_ScpiPackage

### 1.1.1 Version history

Latest release notes summary: Update for FW 5.10

**Version 5.10.121**

- Update for FW 5.10

**Version 5.0.123**

- Update for FW 5.0

**Version 4.70.300.19**

- Fixed bug in interfaces with the name 'base', new docu format

**Version 4.70.300.16**

- Fixed several misspelled arguments and command headers

**Version 4.70.300.15**

- Complete rework of the Repeated capabilities. Before, the driver used extensively the RepCaps Channel, Stream, Subframe, User, Group. Now, they have more fitting names, and also proper ranges and default values.
- All the repcaps ending with Null have ranges starting with 0. 0 is also their default value. For example, ChannelNull starts from 0, while Channel starts from 1. Since this is a breaking change, please make sure your code written in the previous version of the driver is compatible with this new version. This change was necessary in order to assure all the possible settings.

**Version 4.70.205.9**

- Added Documentation
- Added method RsSmab.list_resources()

**Version 4.70.205.8**

- Default HwInterface repcap is 0 (empty command suffix)

**Version 4.70.205.7**

- Second build, fixed enum names

**Version 4.70.205.1**

- First released version

CHAPTER

TWO

# GETTING STARTED

## 2.1 Introduction


**RsSmab** 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 RsSmab example:

```
"""Getting started - how to work with RsSmab Python package.
This example performs basic RF settings on an SMA100B instrument.
It shows the RsSmab calls and their corresponding SCPI commands.
Notice that the python RsSmab interfaces track the SCPI commands syntax."""

from RsSmab import *

# Open the session
smab = RsSmab('TCPIP::10.112.1.67::HISLIP')
# Greetings, stranger...
print(f'Hello, I am: {smab.utilities.idn_string}')

#   OUTPut:STATe ON
smab.output.state.set_value(True)

#   SOURce:FREQuency:MODE CW
smab.source.frequency.set_mode(enums.FreqMode.CW)
```

(continues on next page)

(continued from previous page)

```
#   SOURce:POWer:LEVel:IMMediate:AMPLitude -20
smab.source.power.level.immediate.set_amplitude(-20)

#   SOURce:FREQuency:FIXed 223000000
smab.source.frequency.fixed.set_value(223E6)

#         SOURce:POWer:PEP?
pep = smab.source.power.get_pep()
print(f'PEP level: {pep} dBm')


# Close the session
smab.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

RsSmab 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 RsSmab`

### 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 `RsSmab` 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 RsSmab offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsSmab needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsSmab package to your computer from the pypi.org: https://pypi.org/project/RsSmab/#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\RsSmab-5.10.121.23.tar`

## 2.3 Finding Available Instruments

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

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

from RsSmab import *

# Use the instr_list string items as resource names in the RsSmab constructor
```

(continues on next page)

(continued from previous page)

```
instr_list = RsSmab.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
"""

from RsSmab 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 = RsSmab.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

RsSmab 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 RsSmab object. Below, is a simple Hello World example. Different resource names are examples for different physical interfaces.

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

- Installed RsSmab Python module Version 5.10.121 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsSmab import *

# A good practice is to assure that you have a certain minimum version installed
```

(continues on next page)

(continued from previous page)

```
RsSmab.assert_minimum_version('5.10.121')
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
driver = RsSmab(resource_string_1)

idn = driver.utilities.query_str('*IDN?')
print(f"\nHello, I am: '{idn}'")
print(f'RsSmab 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; RsSmab 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 = RsSmab('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 RsSmab 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 RsSmab allows you to choose which VISA to use:

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

from RsSmab import *

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

idn = driver.utilities.query_str('*IDN?')
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, RsSmab has something for you too - **no Visa installation raw LAN socket**:

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

from RsSmab import *

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

More `option_string` tokens are separated by comma:

```
driver = RsSmab('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 RsSmab objects:

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

from RsSmab import *

driver1 = RsSmab('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsSmab.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 RsSmab 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 RsSmab import *

driver = RsSmab('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 RsSmab import *

driver = RsSmab('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 RsSmab raises an exception. Speaking of exceptions, an important feature of the RsSmab 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 RsSmab import *

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

RsSmab 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 RsSmab 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 RsSmab import *

driver = None
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsSmab('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)
```

(continues on next page)

(continued from previous page)

```
# 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?')

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 RsSmab exceptions
    print(e.args[0])
    print('Some other RsSmab 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 RsSmab, 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 RsSmab 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 RsSmab has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

RsSmab 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 RsSmab 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 = RsSmab('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 RsSmab 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',
    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, RsSmab 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 RsSmab object
"""

import threading
from RsSmab import *


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


driver = RsSmab('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')
```

(continues on next page)

(continued from previous page)

```
# 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 RsSmab objects with shared session
"""

import threading
from RsSmab import *


def execute(session: RsSmab, 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 = RsSmab('TCPIP::192.168.56.101::INSTR')
driver2 = RsSmab.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()
```

(continues on next page)

(continued from previous page)

```
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 RsSmab takes care of it for you. The text below describes this scenario.

Run the following example:

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

import threading
from RsSmab import *


def execute(session: RsSmab, 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 = RsSmab('TCPIP::192.168.56.101::INSTR')
driver2 = RsSmab('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,))
```

(continues on next page)

(continued from previous page)

```
    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 RsSmab import *

driver = RsSmab('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
```

(continues on next page)

(continued from previous page)

```
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 = RsSmab('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 RsSmab import *

driver = RsSmab('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
```

(continues on next page)

(continued from previous page)

```
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 RsSmab import *

driver = RsSmab('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')
```

(continues on next page)

(continued from previous page)

```
# 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 AcDc

```
# Example value:
value = enums.AcDc.AC
# All values (2x):
AC | DC
```

## 3.2 AlcOffMode

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

## 3.3 AlcOnOffAuto

```
# First value:
value = enums.AlcOnOffAuto._0
# Last value:
value = enums.AlcOnOffAuto.PRESet
# All values (9x):
_0 | _1 | AUTO | OFF | OFFTable | ON | ONSample | ONTable
PRESet
```

## 3.4 AmMode

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

## 3.5 AmType

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

## 3.6 AutoManStep

```
# Example value:
value = enums.AutoManStep.AUTO
# All values (3x):
AUTO | MANual | STEP
```

## 3.7 AutoManualMode

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

## 3.8 AutoStep

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

## 3.9 AvionicCarrFreqMode

```
# Example value:
value = enums.AvionicCarrFreqMode.DECimal
# All values (3x):
DECimal | ICAO | USER
```

## 3.10 AvionicCarrFreqModeMrkBcn

```
# Example value:
value = enums.AvionicCarrFreqModeMrkBcn.PREDefined
# All values (2x):
PREDefined | USER
```

## 3.11 AvionicComIdTimeSchem

```
# Example value:
value = enums.AvionicComIdTimeSchem.STD
# All values (2x):
STD | USER
```

## 3.12 AvionicDdmStep

```
# Example value:
value = enums.AvionicDdmStep.DECimal
# All values (2x):
DECimal | PREDefined
```

## 3.13 AvionicExtAm

```
# Example value:
value = enums.AvionicExtAm.EXT
# All values (2x):
EXT | INT
```

## 3.14 AvionicIlsDdmCoup

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

## 3.15 AvionicIlsDdmPol

```
# Example value:
value = enums.AvionicIlsDdmPol.P150_90
# All values (2x):
P150_90 | P90_150
```

## 3.16 AvionicIlsGsMode

```
# Example value:
value = enums.AvionicIlsGsMode.LLOBe
# All values (3x):
LLOBe | NORM | ULOBe
```

## 3.17 AvionicIlsIcaoChan

```
# First value:
value = enums.AvionicIlsIcaoChan.CH18X
# Last value:
value = enums.AvionicIlsIcaoChan.CH56Y
# All values (40x):
CH18X | CH18Y | CH20X | CH20Y | CH22X | CH22Y | CH24X | CH24Y
CH26X | CH26Y | CH28X | CH28Y | CH30X | CH30Y | CH32X | CH32Y
CH34X | CH34Y | CH36X | CH36Y | CH38X | CH38Y | CH40X | CH40Y
CH42X | CH42Y | CH44X | CH44Y | CH46X | CH46Y | CH48X | CH48Y
CH50X | CH50Y | CH52X | CH52Y | CH54X | CH54Y | CH56X | CH56Y
```

## 3.18 AvionicIlsLocMode

```
# Example value:
value = enums.AvionicIlsLocMode.LLOBe
# All values (3x):
LLOBe | NORM | RLOBe
```

## 3.19 AvionicIlsType

```
# Example value:
value = enums.AvionicIlsType.GS
# All values (4x):
GS | GSLope | LOCalize | MBEacon
```

## 3.20 AvionicKnobStep

```
# Example value:
value = enums.AvionicKnobStep.DECimal
# All values (2x):
DECimal | ICAO
```

## 3.21 AvionicVorDir

```
# Example value:
value = enums.AvionicVorDir.FROM
# All values (2x):
FROM | TO
```

## 3.22 AvionicVorIcaoChan

```
# First value:
value = enums.AvionicVorIcaoChan.CH100X
# Last value:
value = enums.AvionicVorIcaoChan.CH99Y
# All values (160x):
CH100X | CH100Y | CH101X | CH101Y | CH102X | CH102Y | CH103X | CH103Y
CH104X | CH104Y | CH105X | CH105Y | CH106X | CH106Y | CH107X | CH107Y
CH108X | CH108Y | CH109X | CH109Y | CH110X | CH110Y | CH111X | CH111Y
CH112X | CH112Y | CH113X | CH113Y | CH114X | CH114Y | CH115X | CH115Y
CH116X | CH116Y | CH117X | CH117Y | CH118X | CH118Y | CH119X | CH119Y
CH120X | CH120Y | CH121X | CH121Y | CH122X | CH122Y | CH123X | CH123Y
CH124X | CH124Y | CH125X | CH125Y | CH126X | CH126Y | CH17X | CH17Y
CH19X | CH19Y | CH21X | CH21Y | CH23X | CH23Y | CH25X | CH25Y
CH27X | CH27Y | CH29X | CH29Y | CH31X | CH31Y | CH33X | CH33Y
CH35X | CH35Y | CH37X | CH37Y | CH39X | CH39Y | CH41X | CH41Y
CH43X | CH43Y | CH45X | CH45Y | CH47X | CH47Y | CH49X | CH49Y
CH51X | CH51Y | CH53X | CH53Y | CH55X | CH55Y | CH57X | CH57Y
CH58X | CH58Y | CH59X | CH59Y | CH70X | CH70Y | CH71X | CH71Y
CH72X | CH72Y | CH73X | CH73Y | CH74X | CH74Y | CH75X | CH75Y
CH76X | CH76Y | CH77X | CH77Y | CH78X | CH78Y | CH79X | CH79Y
CH80X | CH80Y | CH81X | CH81Y | CH82X | CH82Y | CH83X | CH83Y
CH84X | CH84Y | CH85X | CH85Y | CH86X | CH86Y | CH87X | CH87Y
CH88X | CH88Y | CH89X | CH89Y | CH90X | CH90Y | CH91X | CH91Y
CH92X | CH92Y | CH93X | CH93Y | CH94X | CH94Y | CH95X | CH95Y
CH96X | CH96Y | CH97X | CH97Y | CH98X | CH98Y | CH99X | CH99Y
```

## 3.23 AvionicVorMode

```
# Example value:
value = enums.AvionicVorMode.FMSubcarrier
# All values (4x):
FMSubcarrier | NORM | SUBCarrier | VAR
```

## 3.24 ByteOrder

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

## 3.25 CalAdjMode

```
# Example value:
value = enums.CalAdjMode.BURNin
# All values (2x):
BURNin | FULL
```

## 3.26 CalDataMode

```
# Example value:
value = enums.CalDataMode.CUSTomer
# All values (2x):
CUSTomer | FACTory
```

## 3.27 CalDataUpdate

```
# Example value:
value = enums.CalDataUpdate.BBFRC
# All values (6x):
BBFRC | FREQuency | IALL | LEVel | LEVForced | RFFRC
```

## 3.28 CalPowActorLinMode

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

## 3.29 CalPowAmpDetMode

```
# First value:
value = enums.CalPowAmpDetMode.AMP
# Last value:
value = enums.CalPowAmpDetMode.OPU
# All values (13x):
AMP | AT20 | AT40 | AT6 | ATT | AUTO | FIXed | HP
HP6 | OP20 | OP40 | OP6 | OPU
```

## 3.30 CalPowAttMode

```
# Example value:
value = enums.CalPowAttMode.NEW
# All values (2x):
NEW | OLD
```

## 3.31 CalPowBandwidth

```
# Example value:
value = enums.CalPowBandwidth.AUTO
# All values (3x):
AUTO | HIGH | LOW
```

## 3.32 CalPowDetLinMode

```
# Example value:
value = enums.CalPowDetLinMode.AUTO
# All values (4x):
AUTO | OFF | USER1 | USER2
```

## 3.33 CalPowOpuLconMode

```
# Example value:
value = enums.CalPowOpuLconMode.AM
# All values (6x):
AM | AUTO | CW | DAM | USER1 | USER2
```

## 3.34 ClkSynOutType

```
# Example value:
value = enums.ClkSynOutType.CMOS
# All values (4x):
CMOS | DSINe | DSQuare | SESine
```

## 3.35 Colour

```
# Example value:
value = enums.Colour.GREen
# All values (4x):
GREen | NONE | RED | YELLow
```

## 3.36 DecimalSeparator

```
# Example value:
value = enums.DecimalSeparator.COMMa
# All values (2x):
COMMa | DOT
```

## 3.37 DevExpFormat

```
# Example value:
value = enums.DevExpFormat.CGPRedefined
# All values (4x):
CGPRedefined | CGUSer | SCPI | XML
```

## 3.38 DexchExtension

```
# Example value:
value = enums.DexchExtension.CSV
# All values (2x):
CSV | TXT
```

## 3.39 DexchMode

```
# Example value:
value = enums.DexchMode.EXPort
# All values (2x):
EXPort | IMPort
```

## 3.40 DexchSepCol

```
# Example value:
value = enums.DexchSepCol.COMMa
# All values (4x):
COMMa | SEMicolon | SPACe | TABulator
```

## 3.41 DiagBgColor

```
# Example value:
value = enums.DiagBgColor.BLACk
# All values (2x):
BLACk | WHITe
```

## 3.42 DispKeybLockMode

```
# Example value:
value = enums.DispKeybLockMode.DISabled
# All values (5x):
DISabled | DONLy | ENABled | TOFF | VNConly
```

## 3.43 ErFpowSensMapping

```
# First value:
value = enums.ErFpowSensMapping.SENS1
# Last value:
value = enums.ErFpowSensMapping.UNMapped
# All values (9x):
SENS1 | SENS2 | SENS3 | SENS4 | SENSor1 | SENSor2 | SENSor3 | SENSor4
UNMapped
```

## 3.44 FilterWidth

```
# Example value:
value = enums.FilterWidth.NARRow
# All values (2x):
NARRow | WIDE
```

## 3.45 FmMode

```
# Example value:
value = enums.FmMode.HBANdwidth
# All values (2x):
HBANdwidth | LNOise
```

## 3.46 FmSour

```
# Example value:
value = enums.FmSour.EXT1
# All values (7x):
EXT1 | EXT2 | EXTernal | INTernal | LF1 | LF2 | NOISe
```

## 3.47 FormData

```
# Example value:
value = enums.FormData.ASCii
# All values (2x):
ASCii | PACKed
```

## 3.48 FormStatReg

```
# Example value:
value = enums.FormStatReg.ASCii
# All values (4x):
ASCii | BINary | HEXadecimal | OCTal
```

## 3.49 FreqMode

```
# Example value:
value = enums.FreqMode.COMBined
# All values (5x):
COMBined | CW | FIXed | LIST | SWEep
```

## 3.50 FreqPllModeF

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

## 3.51 FreqStepMode

```
# Example value:
value = enums.FreqStepMode.DECimal
# All values (2x):
DECimal | USER
```

## 3.52 FreqSweepType

```
# Example value:
value = enums.FreqSweepType.ANALog
# All values (2x):
ANALog | STEPped
```

## 3.53 FrontPanelLayout

```
# Example value:
value = enums.FrontPanelLayout.DIGits
# All values (2x):
DIGits | LETTers
```

## 3.54 HardCopyImageFormat

```
# Example value:
value = enums.HardCopyImageFormat.BMP
# All values (4x):
BMP | JPG | PNG | XPM
```

## 3.55 HardCopyRegion

```
# Example value:
value = enums.HardCopyRegion.ALL
# All values (2x):
ALL | DIALog
```

## 3.56 HcopyDestination

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

## 3.57 IecDevId

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

## 3.58 IecTermMode

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

## 3.59 Imp

```
# Example value:
value = enums.Imp.G50
# All values (3x):
G50 | G600 | HIGH
```

## 3.60 InclExcl

```
# Example value:
value = enums.InclExcl.EXCLude
# All values (2x):
EXCLude | INCLude
```

## 3.61 InpImpRf

```
# Example value:
value = enums.InpImpRf.G10K
# All values (3x):
G10K | G1K | G50
```

## 3.62 KbLayout

```
# First value:
value = enums.KbLayout.CHINese
# Last value:
value = enums.KbLayout.SWEDish
# All values (20x):
CHINese | DANish | DUTBe | DUTCh | ENGLish | ENGUK | ENGUS | FINNish
FREBe | FRECa | FRENch | GERMan | ITALian | JAPanese | KORean | NORWegian
PORTuguese | RUSSian | SPANish | SWEDish
```

## 3.63 LeftRightDirection

```
# Example value:
value = enums.LeftRightDirection.LEFT
# All values (2x):
LEFT | RIGHt
```

## 3.64 LfBwidth

```
# Example value:
value = enums.LfBwidth.BW0M2
# All values (2x):
BW0M2 | BW10m
```

## 3.65 LfFreqMode

```
# Example value:
value = enums.LfFreqMode.CW
# All values (3x):
CW | FIXed | SWEep
```

## 3.66 LfShapeBfAmily

```
# Example value:
value = enums.LfShapeBfAmily.PULSe
# All values (5x):
PULSe | SINE | SQUare | TRAPeze | TRIangle
```

## 3.67 LfSource

```
# First value:
value = enums.LfSource.AM
# Last value:
value = enums.LfSource.NOISe
# All values (17x):
AM | AMA | AMB | EXT1 | EXT2 | FMPM | FMPMA | FMPMB
LF1 | LF1A | LF1B | LF2 | LF2A | LF2B | NOISA | NOISB
NOISe
```

## 3.68 LfSweepSource

```
# Example value:
value = enums.LfSweepSource.LF1
# All values (2x):
LF1 | LF2
```

## 3.69 LmodRunMode

```
# Example value:
value = enums.LmodRunMode.LEARned
# All values (2x):
LEARned | LIVE
```

## 3.70 LowHigh

```
# Example value:
value = enums.LowHigh.HIGH
# All values (2x):
HIGH | LOW
```

## 3.71 MeasRespHcOpCsvcLmSep

```
# Example value:
value = enums.MeasRespHcOpCsvcLmSep.BLANk
# All values (4x):
BLANk | COMMa | SEMicolon | TABulator
```

## 3.72 MeasRespHcOpCsvhEader

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

## 3.73 MeasRespHcOpCsvoRient

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

## 3.74 MeasRespHcOpFileFormat

```
# Example value:
value = enums.MeasRespHcOpFileFormat.BMP
# All values (5x):
BMP | CSV | JPG | PNG | XPM
```

## 3.75 MeasRespMath

```
# First value:
value = enums.MeasRespMath.T1REf
# Last value:
value = enums.MeasRespMath.T4T4
# All values (20x):
T1REf | T1T1 | T1T2 | T1T3 | T1T4 | T2REf | T2T1 | T2T2
T2T3 | T2T4 | T3REf | T3T1 | T3T2 | T3T3 | T3T4 | T4REf
T4T1 | T4T2 | T4T3 | T4T4
```

## 3.76 MeasRespMode

```
# Example value:
value = enums.MeasRespMode.FREQuency
# All values (3x):
FREQuency | POWer | TIME
```

## 3.77 MeasRespPulsThrBase

```
# Example value:
value = enums.MeasRespPulsThrBase.POWer
# All values (2x):
POWer | VOLTage
```

## 3.78 MeasRespSpacingMode

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

## 3.79 MeasRespTimeAverage

```
# First value:
value = enums.MeasRespTimeAverage._1
# Last value:
value = enums.MeasRespTimeAverage._8
# All values (11x):
_1 | _1024 | _128 | _16 | _2 | _256 | _32 | _4
_512 | _64 | _8
```

## 3.80 MeasRespTimeGate

```
# Example value:
value = enums.MeasRespTimeGate.TRAC1
# All values (8x):
TRAC1 | TRAC2 | TRAC3 | TRAC4 | TRACe1 | TRACe2 | TRACe3 | TRACe4
```

## 3.81 MeasRespTimingMode

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

## 3.82 MeasRespTraceColor

```
# Example value:
value = enums.MeasRespTraceColor.BLUE
# All values (7x):
BLUE | GRAY | GREen | INVers | MAGenta | RED | YELLow
```

## 3.83 MeasRespTraceCopyDest

```
# Example value:
value = enums.MeasRespTraceCopyDest.REFerence
# All values (1x):
REFerence
```

## 3.84 MeasRespTraceFeed

```
# First value:
value = enums.MeasRespTraceFeed.NONE
# Last value:
value = enums.MeasRespTraceFeed.SENSor4
# All values (10x):
NONE | REFerence | SENS1 | SENS2 | SENS3 | SENS4 | SENSor1 | SENSor2
SENSor3 | SENSor4
```

## 3.85 MeasRespTraceState

```
# Example value:
value = enums.MeasRespTraceState.HOLD
# All values (3x):
HOLD | OFF | ON
```

## 3.86 MeasRespTrigAutoSet

```
# Example value:
value = enums.MeasRespTrigAutoSet.ONCE
# All values (1x):
ONCE
```

## 3.87 MeasRespTrigMode

```
# Example value:
value = enums.MeasRespTrigMode.AUTO
# All values (4x):
AUTO | EXTernal | FREE | INTernal
```

## 3.88 MeasRespYsCaleEvents

```
# Example value:
value = enums.MeasRespYsCaleEvents.AND
# All values (2x):
AND | OR
```

## 3.89 MeasRespYsCaleMode

```
# Example value:
value = enums.MeasRespYsCaleMode.CEXPanding
# All values (5x):
CEXPanding | CFLoating | FEXPanding | FFLoating | OFF
```

## 3.90 ModulationDevMode

```
# Example value:
value = enums.ModulationDevMode.RATio
# All values (3x):
RATio | TOTal | UNCoupled
```

## 3.91 NetMode

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

## 3.92 NoisDistrib

```
# Example value:
value = enums.NoisDistrib.EQUal
# All values (4x):
EQUal | GAUSs | NORMal | UNIForm
```

## 3.93 NormalInverted

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

## 3.94 ParameterSetMode

```
# Example value:
value = enums.ParameterSetMode.GLOBal
# All values (2x):
GLOBal | LIST
```

## 3.95 Parity

```
# Example value:
value = enums.Parity.EVEN
# All values (3x):
EVEN | NONE | ODD
```

## 3.96 PixelTestPredefined

```
# First value:
value = enums.PixelTestPredefined.AUTO
# Last value:
value = enums.PixelTestPredefined.WHITe
# All values (9x):
AUTO | BLACk | BLUE | GR25 | GR50 | GR75 | GREen | RED
WHITe
```

## 3.97 PmMode

```
# Example value:
value = enums.PmMode.HBANdwidth
# All values (3x):
HBANdwidth | HDEViation | LNOise
```

## 3.98 PowAlcDetSensitivity

```
# Example value:
value = enums.PowAlcDetSensitivity.AUTO
# All values (5x):
AUTO | FIXed | HIGH | LOW | MEDium
```

## 3.99 PowAlcStateWithExtAlc

```
# First value:
value = enums.PowAlcStateWithExtAlc._0
# Last value:
value = enums.PowAlcStateWithExtAlc.PRESet
# All values (10x):
_0 | _1 | AUTO | EALC | OFF | OFFTable | ON | ONSample
ONTable | PRESet
```

## 3.100 PowAttMode

```
# Example value:
value = enums.PowAttMode.AUTO
# All values (5x):
AUTO | FIXed | HPOWer | MANual | NORMal
```

## 3.101 PowAttModeOut

```
# Example value:
value = enums.PowAttModeOut.AUTO
# All values (4x):
AUTO | FIXed | HPOWer | NORMal
```

## 3.102 PowAttRfOffMode

```
# Example value:
value = enums.PowAttRfOffMode.FATTenuation
# All values (2x):
FATTenuation | UNCHanged
```

## 3.103 PowAttStepArt

```
# Example value:
value = enums.PowAttStepArt.ELECtronic
# All values (2x):
ELECtronic | MECHanical
```

## 3.104 PowCntrlSelect

```
# Example value:
value = enums.PowCntrlSelect.SENS1
# All values (8x):
SENS1 | SENS2 | SENS3 | SENS4 | SENSor1 | SENSor2 | SENSor3 | SENSor4
```

## 3.105 PowHarmMode

```
# Example value:
value = enums.PowHarmMode._1
# All values (3x):
_1 | AUTO | ON
```

## 3.106 PowLevBehaviour

```
# Example value:
value = enums.PowLevBehaviour.AUTO
# All values (7x):
AUTO | CPHase | CVSWr | HDUN | MONotone | UNINterrupted | USER
```

## 3.107 PowLevMode

```
# Example value:
value = enums.PowLevMode.LOWDistortion
# All values (3x):
LOWDistortion | LOWNoise | NORMal
```

## 3.108 PowSensDisplayPriority

```
# Example value:
value = enums.PowSensDisplayPriority.AVERage
# All values (2x):
AVERage | PEAK
```

## 3.109 PowSensFiltType

```
# Example value:
value = enums.PowSensFiltType.AUTO
# All values (3x):
AUTO | NSRatio | USER
```

## 3.110 PowSensSource

```
# Example value:
value = enums.PowSensSource.A
# All values (4x):
A | B | RF | USER
```

## 3.111 PulsMode

```
# Example value:
value = enums.PulsMode.DOUBle
# All values (4x):
DOUBle | PHOPptrain | PTRain | SINGle
```

## 3.112 PulsTransType

```
# Example value:
value = enums.PulsTransType.FAST
# All values (2x):
FAST | SMOothed
```

## 3.113 PulsTrigModeWithSingle

```
# Example value:
value = enums.PulsTrigModeWithSingle.AUTO
# All values (5x):
AUTO | EGATe | ESINgle | EXTernal | SINGle
```

## 3.114 RecScpiCmdMode

```
# Example value:
value = enums.RecScpiCmdMode.AUTO
# All values (4x):
AUTO | DAUTo | MANual | OFF
```

## 3.115 RepeatMode

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

## 3.116 RfFreqMultCcorMode

```
# Example value:
value = enums.RfFreqMultCcorMode.HPRecision
# All values (3x):
HPRecision | NONE | STANdard
```

## 3.117 Rosc1GoUtpFreqMode

```
# Example value:
value = enums.Rosc1GoUtpFreqMode.DER1G
# All values (3x):
DER1G | LOOPthrough | OFF
```

## 3.118 RoscFreqExt

```
# Example value:
value = enums.RoscFreqExt._100MHZ
# All values (4x):
_100MHZ | _10MHZ | _1GHZ | VARiable
```

## 3.119 RoscOutpFreqMode

```
# Example value:
value = enums.RoscOutpFreqMode.DER100M
# All values (4x):
DER100M | DER10M | LOOPthrough | OFF
```

## 3.120 Rs232BdRate

```
# Example value:
value = enums.Rs232BdRate._115200
# All values (7x):
_115200 | _19200 | _2400 | _38400 | _4800 | _57600 | _9600
```

## 3.121 Rs232StopBits

```
# Example value:
value = enums.Rs232StopBits._1
# All values (2x):
_1 | _2
```

## 3.122 SelftLev

```
# Example value:
value = enums.SelftLev.CUSTomer
# All values (3x):
CUSTomer | PRODuction | SERVice
```

## 3.123 SelftLevWrite

```
# Example value:
value = enums.SelftLevWrite.CUSTomer
# All values (4x):
CUSTomer | NONE | PRODuction | SERVice
```

## 3.124 SelOutpMarkUser

```
# Example value:
value = enums.SelOutpMarkUser.MARK
# All values (2x):
MARK | USER
```

## 3.125 SelOutpVxAxis

```
# Example value:
value = enums.SelOutpVxAxis.S0V25
# All values (4x):
S0V25 | S0V5 | S1V0 | XAXis
```

## 3.126 SensorModeAll

```
# Example value:
value = enums.SensorModeAll.AUTO
# All values (3x):
AUTO | EXTSingle | SINGle
```

## 3.127 SingExtAuto

```
# Example value:
value = enums.SingExtAuto.AUTO
# All values (8x):
AUTO | BUS | DHOP | EAUTo | EXTernal | HOP | IMMediate | SINGle
```

## 3.128 SlopeType

```
# Example value:
value = enums.SlopeType.NEGative
# All values (2x):
NEGative | POSitive
```

## 3.129 SourceInt

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

## 3.130 Spacing

```
# Example value:
value = enums.Spacing.LINear
# All values (3x):
LINear | LOGarithmic | RAMP
```

## 3.131 StagMode

```
# Example value:
value = enums.StagMode.AUTO
# All values (3x):
AUTO | FIXed | USER
```

## 3.132 StateExtended

```
# Example value:
value = enums.StateExtended._0
# All values (6x):
_0 | _1 | _2 | DEFault | OFF | ON
```

## 3.133 SweCyclMode

```
# Example value:
value = enums.SweCyclMode.SAWTooth
# All values (2x):
SAWTooth | TRIangle
```

## 3.134 SweepType

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

## 3.135 SweMarkActive

```
# First value:
value = enums.SweMarkActive.M01
# Last value:
value = enums.SweMarkActive.NONE
# All values (11x):
M01 | M02 | M03 | M04 | M05 | M06 | M07 | M08
M09 | M10 | NONE
```

## 3.136 Test

```
# Example value:
value = enums.Test._0
# All values (4x):
_0 | _1 | RUNning | STOPped
```

## 3.137 TestCalSelected

```
# Example value:
value = enums.TestCalSelected._0
# All values (2x):
_0 | _1
```

## 3.138 TimeProtocol

```
# Example value:
value = enums.TimeProtocol._0
# All values (6x):
_0 | _1 | NONE | NTP | OFF | ON
```

## 3.139 TraceSourceAll

```
# Example value:
value = enums.TraceSourceAll.HOLD
# All values (8x):
HOLD | OFF | ON | REF | SEN1 | SEN2 | SEN3 | SEN4
```

## 3.140 TrigSweepImmBusExt

```
# Example value:
value = enums.TrigSweepImmBusExt.BUS
# All values (3x):
BUS | EXTernal | IMMediate
```

## 3.141 TrigSweepSourNoHopExtAuto

```
# Example value:
value = enums.TrigSweepSourNoHopExtAuto.AUTO
# All values (5x):
AUTO | BUS | EXTernal | IMMediate | SINGle
```

## 3.142 UnchOff

```
# Example value:
value = enums.UnchOff.OFF
# All values (2x):
OFF | UNCHanged
```

## 3.143 UnitAngle

```
# Example value:
value = enums.UnitAngle.DEGree
# All values (3x):
DEGree | DEGRee | RADian
```

## 3.144 UnitPower

```
# Example value:
value = enums.UnitPower.DBM
# All values (3x):
DBM | DBUV | V
```

## 3.145 UnitPowSens

```
# Example value:
value = enums.UnitPowSens.DBM
# All values (3x):
DBM | DBUV | WATT
```

## 3.146 UnitSpeed

```
# Example value:
value = enums.UnitSpeed.KMH
# All values (4x):
KMH | MPH | MPS | NMPH
```

## 3.147 UpDownDirection

```
# Example value:
value = enums.UpDownDirection.DOWN
# All values (2x):
DOWN | UP
```

## 3.148 UpdPolicyMode

```
# Example value:
value = enums.UpdPolicyMode.CONFirm
# All values (3x):
CONFirm | IGNore | STRict
```

CHAPTER

# FOUR

# REPCAPS

## 4.1 HwInstance (Global)

```
# Setting:
driver.repcap_hwInstance_set(repcap.HwInstance.InstA)
# Range:
InstA .. InstH
# All values (8x):
InstA | InstB | InstC | InstD | InstE | InstF | InstG | InstH
```

## 4.2 BitNumberNull

```
# First value:
value = repcap.BitNumberNull.Nr0
# Range:
Nr0 .. Nr15
# All values (16x):
Nr0 | Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7
Nr8 | Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15
```

## 4.3 Channel

```
# First value:
value = repcap.Channel.Nr1
# Range:
Nr1 .. Nr64
# All values (64x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23 | Nr24
Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31 | Nr32
Nr33 | Nr34 | Nr35 | Nr36 | Nr37 | Nr38 | Nr39 | Nr40
Nr41 | Nr42 | Nr43 | Nr44 | Nr45 | Nr46 | Nr47 | Nr48
Nr49 | Nr50 | Nr51 | Nr52 | Nr53 | Nr54 | Nr55 | Nr56
Nr57 | Nr58 | Nr59 | Nr60 | Nr61 | Nr62 | Nr63 | Nr64
```

## 4.4 ErrorCount

```
# First value:
value = repcap.ErrorCount.Nr1
# Range:
Nr1 .. Nr16
# All values (16x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
```

## 4.5 Gate

```
# First value:
value = repcap.Gate.Nr1
# Range:
Nr1 .. Nr8
# All values (8x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
```

## 4.6 GeneratorIx

```
# First value:
value = repcap.GeneratorIx.Nr1
# Range:
Nr1 .. Nr8
# All values (8x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
```

## 4.7 InputIx

```
# First value:
value = repcap.InputIx.Nr1
# Range:
Nr1 .. Nr8
# All values (8x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
```

## 4.8 Level

```
# First value:
value = repcap.Level.Nr1
# Range:
Nr1 .. Nr16
# All values (16x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
```

## 4.9 LfOutput

```
# First value:
value = repcap.LfOutput.Nr1
# Values (4x):
Nr1 | Nr2 | Nr3 | Nr4
```

## 4.10 Marker

```
# First value:
value = repcap.Marker.Nr0
# Range:
Nr0 .. Nr31
# All values (32x):
Nr0 | Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7
Nr8 | Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15
Nr16 | Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23
Nr24 | Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31
```

## 4.11 Math

```
# First value:
value = repcap.Math.Nr1
# Range:
Nr1 .. Nr8
# All values (8x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
```

## 4.12 Trace

```
# First value:
value = repcap.Trace.Nr1
# Range:
Nr1 .. Nr32
# All values (32x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23 | Nr24
Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31 | Nr32
```

CHAPTER

# FIVE

# EXAMPLES

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

```
"""Getting started - how to work with RsSmab Python package.
This example performs basic RF settings on an SMA100B instrument.
It shows the RsSmab calls and their corresponding SCPI commands.
Notice that the python RsSmab interfaces track the SCPI commands syntax."""

from RsSmab import *

# Open the session
smab = RsSmab('TCPIP::10.112.1.67::HISLIP')
# Greetings, stranger...
print(f'Hello, I am: {smab.utilities.idn_string}')

#   OUTPut:STATe ON
smab.output.state.set_value(True)

#   SOURce:FREQuency:MODE CW
smab.source.frequency.set_mode(enums.FreqMode.CW)

#   SOURce:POWer:LEVel:IMMediate:AMPLitude -20
smab.source.power.level.immediate.set_amplitude(-20)

#   SOURce:FREQuency:FIXed 223000000
smab.source.frequency.fixed.set_value(223E6)

#         SOURce:POWer:PEP?
pep = smab.source.power.get_pep()
print(f'PEP level: {pep} dBm')


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

```
"""Basic example of importing the package, initializing the session and performing basic␣
↪generator settings."""

from RsSmab import *

RsSmab.assert_minimum_version('4.70.300')
```

(continues on next page)

(continued from previous page)

```
smab = RsSmab('TCPIP::10.112.1.64::HISLIP')
# smab = RsSmab('TCPIP::10.112.0.106::5025::SOCKET', options='SelectVisa=SocketIo') # No␣
↪VISA needed
print(f'Driver Info: {smab.utilities.driver_version}')
print(f'Instrument: {smab.utilities.idn_string}')

# Instrument options are properly parsed, and sorted (k-options first)
print(f'Instrument options: {",".join(smab.utilities.instrument_options)}')

# Driver's instrument status checking ( SYST:ERR? ) after each command (default value is␣
↪True):
smab.utilities.instrument_status_checking = True

smab.output.state.set_value(True)
smab.source.frequency.set_mode(enums.FreqMode.CW)
smab.source.power.level.immediate.set_amplitude(-20)
smab.source.frequency.fixed.set_value(223E6)

# You can still use the direct SCPI interface:
response = smab.utilities.query_str('*IDN?')
print(f'Direct SCPI response on *IDN?: {response}')
smab.close()
```

```
"""Example showing how you can transfer a big file to the instrument and from the␣
↪instrument with showing the progress.
Since the SMA100B is quite fast on data transfer, we slow it down by waiting for 100ms␣
↪between each chunk transfer (1MB)
This way we see the transfer progress better and we do not need a file that is so big -␣
↪let's take cca 20MB.
For big files, use the example without the time.sleep(0.1)"""

import time
import numpy as np
from RsSmab import *


def my_transfer_handler(args):
    """Function called each time a chunk of data is transferred"""
    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')
    # Slow down the transfer by 200ms to see the progress better
    time.sleep(0.1)


RsSmab.assert_minimum_version('4.70.300')
```

(continues on next page)

(continued from previous page)

```
smab = RsSmab('TCPIP::10.112.1.64::HISLIP')
print(smab.utilities.idn_string)
smab.utilities.reset()

pc_file = r'c:\temp\bigFile.bin'
instr_file = '/var/user/bigFileInstr.bin'
pc_file_back = r'c:\temp\bigFileBack.bin'

# Generate a random file of 20MB size
x1mb = 1024 * 1024
with open(pc_file, 'wb') as file:
    for x in range(20):
        file.write(np.random.bytes(x1mb))

# Send the file to the instrument with events
smab.events.on_write_handler = my_transfer_handler
smab.utilities.data_chunk_size = x1mb
print(f'Sending file to the instrument...')
smab.utilities.send_file_from_pc_to_instrument(pc_file, instr_file)
smab.events.on_write_handler = None
print(f'Receiving file from the instrument...')
smab.events.on_read_handler = my_transfer_handler
smab.utilities.read_file_from_instrument_to_pc(instr_file, pc_file_back)
smab.events.on_read_handler = None
smab.close()
```

CHAPTER

# SIX

# RSSMAB API STRUCTURE

## Global RepCaps

```
driver = RsSmab('TCPIP::192.168.2.101::hislip0')
# HwInstance range: InstA .. InstH
rc = driver.repcap_hwInstance_get()
driver.repcap_hwInstance_set(repcap.HwInstance.InstA)
```

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

1126 total commands, 25 Subgroups, 0 group commands

Initializes new RsSmab 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`
- `ScpiQuotes = double'. - for SCPI commands, you can define how strings are quoted. With single or double quotes. Possible values: single | double | {char}. Default: ``single`
- `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 RsSmab.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 RsSmab session.

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

Creates a new RsSmab 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

## 6.1 Calculate

**class CalculateCls**

Calculate commands group definition. 24 total commands, 1 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

### 6.1.1 Power

**class PowerCls**

Power commands group definition. 24 total commands, 1 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.1.1.1 Sweep

**class SweepCls**

Sweep commands group definition. 24 total commands, 3 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

##### 6.1.1.1.1 Frequency

**class FrequencyCls**

Frequency 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.calculate.power.sweep.frequency.clone()
```

#### Subgroups

##### 6.1.1.1.1.1 Marker<Marker>

#### RepCap Settings

```
# Range: Nr0 .. Nr31
rc = driver.calculate.power.sweep.frequency.marker.repcap_marker_get()
driver.calculate.power.sweep.frequency.marker.repcap_marker_set(repcap.Marker.Nr0)
```

**class MarkerCls**

Marker commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Marker, default value after init: Marker.Nr0

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.frequency.marker.clone()
```

#### Subgroups

##### 6.1.1.1.1.2 Feed

#### SCPI Command :

```
CALCulate:[POWer]:SWEep:FREQuency:MARKer<CH>:FEED
```

**class FeedCls**

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

**get**(*marker=Marker.Default*) → MeasRespTimeGate

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MARKer<CH>:FEED
value: enums.MeasRespTimeGate = driver.calculate.power.sweep.frequency.marker.
↪feed.get(marker = repcap.Marker.Default)
```

No command help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
marker_binding: No help available

**set**(*marker_binding: MeasRespTimeGate*, *marker=Marker.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MARKer<CH>:FEED
driver.calculate.power.sweep.frequency.marker.feed.set(marker_binding = enums.
↪MeasRespTimeGate.TRAC1, marker = repcap.Marker.Default)
```

No command help available

**param marker_binding**
No help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.1.1.1.1.3 State

#### SCPI Command :

```
CALCulate:[POWer]:SWEep:FREQuency:MARKer<CH>:STATe
```

**class StateCls**

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

**get**(*marker=Marker.Default*) → bool

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MARKer<CH>:STATe
value: bool = driver.calculate.power.sweep.frequency.marker.state.get(marker =␣
↪repcap.Marker.Default)
```

No command help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
marker_state: No help available

**set**(*marker_state: bool*, *marker=Marker.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MARKer<CH>:STATe
driver.calculate.power.sweep.frequency.marker.state.set(marker_state = False,␣
↪marker = repcap.Marker.Default)
```

No command help available

**param marker_state**
No help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.1.1.1.1.4 Math<Math>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.calculate.power.sweep.frequency.math.repcap_math_get()
driver.calculate.power.sweep.frequency.math.repcap_math_set(repcap.Math.Nr1)
```

**class MathCls**

Math commands group definition. 4 total commands, 4 Subgroups, 0 group commands Repeated Capability: Math, default value after init: Math.Nr1

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.frequency.math.clone()
```

##### Subgroups

#### 6.1.1.1.1.5 State

##### SCPI Command :

```
CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:STATe
```

**class StateCls**

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

**get**(*math=Math.Default*) → bool

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:STATe
value: bool = driver.calculate.power.sweep.frequency.math.state.get(math =␣
↪repcap.Math.Default)
```

Activates the trace mathematics mode for 'Frequency' measurement. This feature enables you to calculate the difference between the measurement values of two traces. For further calculation, a math result can also be assigned to a trace.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:STATe
driver.calculate.power.sweep.frequency.math.state.set(state = False, math =␣
↪repcap.Math.Default)
```

Activates the trace mathematics mode for 'Frequency' measurement. This feature enables you to calculate the difference between the measurement values of two traces. For further calculation, a math result can also be assigned to a trace.

**param state**
0| 1| OFF| ON

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.1.6 Subtract

#### SCPI Command :

```
CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:SUBTract
```

**class SubtractCls**

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

**get**(*math=Math.Default*) → MeasRespMath

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:SUBTract
value: enums.MeasRespMath = driver.calculate.power.sweep.frequency.math.
↪subtract.get(math = repcap.Math.Default)
```

Subtracts the operands 1 and 2 and assigns the result to the selected trace in 'Frequency' measurement mode.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
subtract: T1T1| T1T2| T1T3| T1T4| T1REf| T2T1| T2T2| T2T3| T2T4| T2REf| T3T1| T3T2| T3T3| T3T4| T3REf| T4T1| T4T2| T4T3| T4T4| T4REf

**set**(*subtract: MeasRespMath*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:SUBTract
driver.calculate.power.sweep.frequency.math.subtract.set(subtract = enums.
↪MeasRespMath.T1REf, math = repcap.Math.Default)
```

Subtracts the operands 1 and 2 and assigns the result to the selected trace in 'Frequency' measurement mode.

**param subtract**
T1T1| T1T2| T1T3| T1T4| T1REf| T2T1| T2T2| T2T3| T2T4| T2REf| T3T1| T3T2| T3T3| T3T4| T3REf| T4T1| T4T2| T4T3| T4T4| T4REf

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.1.7 Xval

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:XVAL
```

**class XvalCls**

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

**get**(*math=Math.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:XVAL
value: float = driver.calculate.power.sweep.frequency.math.xval.get(math =␣
↪repcap.Math.Default)
```

Sets the x-axis values for calculating the reference curve in frequency measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
xval: float Range: 0 to 1E12

**set**(*xval: float*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:XVAL
driver.calculate.power.sweep.frequency.math.xval.set(xval = 1.0, math = repcap.
↪Math.Default)
```

Sets the x-axis values for calculating the reference curve in frequency measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param xval**
float Range: 0 to 1E12

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.1.8 Yval

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:YVAL
```

**class YvalCls**

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

**get**(*math=Math.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:YVAL
value: float = driver.calculate.power.sweep.frequency.math.yval.get(math =␣
↪repcap.Math.Default)
```

Sets the y-axis values for calculating the reference curve in frequency measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
yval: float Range: 200 to 100

**set**(*yval: float*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:FREQuency:MATH<CH>:YVAL
driver.calculate.power.sweep.frequency.math.yval.set(yval = 1.0, math = repcap.
↪Math.Default)
```

Sets the y-axis values for calculating the reference curve in frequency measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param yval**
float Range: 200 to 100

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.2 Power

**class PowerCls**

Power 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.calculate.power.sweep.power.clone()
```

### Subgroups

#### 6.1.1.1.2.1 Marker<Marker>

### RepCap Settings

```
# Range: Nr0 .. Nr31
rc = driver.calculate.power.sweep.power.marker.repcap_marker_get()
driver.calculate.power.sweep.power.marker.repcap_marker_set(repcap.Marker.Nr0)
```

**class MarkerCls**

Marker commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Marker, default value after init: Marker.Nr0

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.power.marker.clone()
```

#### Subgroups

##### 6.1.1.1.2.2 Feed

#### SCPI Command :

```
CALCulate:[POWer]:SWEep:POWer:MARKer<CH>:FEED
```

**class FeedCls**

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

**get**(*marker=Marker.Default*) → MeasRespTimeGate

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MARKer<CH>:FEED
value: enums.MeasRespTimeGate = driver.calculate.power.sweep.power.marker.feed.
↪get(marker = repcap.Marker.Default)
```

No command help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
marker_binding: No help available

**set**(*marker_binding: MeasRespTimeGate*, *marker=Marker.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MARKer<CH>:FEED
driver.calculate.power.sweep.power.marker.feed.set(marker_binding = enums.
↪MeasRespTimeGate.TRAC1, marker = repcap.Marker.Default)
```

No command help available

**param marker_binding**
No help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.1.1.1.2.3 State

##### SCPI Command :

```
CALCulate:[POWer]:SWEep:POWer:MARKer<CH>:STATe
```

**class StateCls**

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

**get**(*marker=Marker.Default*) → bool

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MARKer<CH>:STATe
value: bool = driver.calculate.power.sweep.power.marker.state.get(marker =␣
↪repcap.Marker.Default)
```

No command help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
marker_state: No help available

**set**(*marker_state: bool*, *marker=Marker.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MARKer<CH>:STATe
driver.calculate.power.sweep.power.marker.state.set(marker_state = False,␣
↪marker = repcap.Marker.Default)
```

No command help available

**param marker_state**
No help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.1.1.1.2.4 Math<Math>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.calculate.power.sweep.power.math.repcap_math_get()
driver.calculate.power.sweep.power.math.repcap_math_set(repcap.Math.Nr1)
```

**class MathCls**

Math commands group definition. 4 total commands, 4 Subgroups, 0 group commands Repeated Capability: Math, default value after init: Math.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.power.math.clone()
```

### Subgroups

#### 6.1.1.1.2.5 State

### SCPI Command :

```
CALCulate:[POWer]:SWEep:POWer:MATH<CH>:STATe
```

**class StateCls**

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

**get**(*math=Math.Default*) → bool

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:STATe
value: bool = driver.calculate.power.sweep.power.math.state.get(math = repcap.
↪Math.Default)
```

Activates the trace mathematics mode for 'Power' measurement. This feature enables you to calculate the difference between the measurement values of two traces. For further calculation, a math result can also be assigned to a trace.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:STATe
driver.calculate.power.sweep.power.math.state.set(state = False, math = repcap.
↪Math.Default)
```

Activates the trace mathematics mode for 'Power' measurement. This feature enables you to calculate the difference between the measurement values of two traces. For further calculation, a math result can also be assigned to a trace.

**param state**
0| 1| OFF| ON

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.2.6 Subtract

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:POWer:MATH<CH>:SUBTract
```

**class SubtractCls**

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

**get**(*math=Math.Default*) → MeasRespMath

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:SUBTract
value: enums.MeasRespMath = driver.calculate.power.sweep.power.math.subtract.
↪get(math = repcap.Math.Default)
```

Subtracts the operands 1 and 2 and assigns the result to the selected trace in 'Power' measurement mode.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
subtract: T1T1| T1T2| T1T3| T1T4| T1REf| T2T1| T2T2| T2T3| T2T4| T2REf| T3T1| T3T2| T3T3| T3T4| T3REf| T4T1| T4T2| T4T3| T4T4| T4REf

**set**(*subtract: MeasRespMath*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:SUBTract
driver.calculate.power.sweep.power.math.subtract.set(subtract = enums.
↪MeasRespMath.T1REf, math = repcap.Math.Default)
```

Subtracts the operands 1 and 2 and assigns the result to the selected trace in 'Power' measurement mode.

**param subtract**
T1T1| T1T2| T1T3| T1T4| T1REf| T2T1| T2T2| T2T3| T2T4| T2REf| T3T1| T3T2| T3T3| T3T4| T3REf| T4T1| T4T2| T4T3| T4T4| T4REf

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.2.7 Xval

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:POWer:MATH<CH>:XVAL
```

**class XvalCls**

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

**get**(*math=Math.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:XVAL
value: float = driver.calculate.power.sweep.power.math.xval.get(math = repcap.
↪Math.Default)
```

Sets the x-axis values for calculating the reference curve in power measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
xval: float Range: -145 to 20

**set**(*xval: float*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:XVAL
driver.calculate.power.sweep.power.math.xval.set(xval = 1.0, math = repcap.Math.
↪Default)
```

Sets the x-axis values for calculating the reference curve in power measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param xval**
float Range: -145 to 20

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.2.8 Yval

##### SCPI Command :

```
CALCulate:[POWer]:SWEep:POWer:MATH<CH>:YVAL
```

**class YvalCls**

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

**get**(*math=Math.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:YVAL
value: float = driver.calculate.power.sweep.power.math.yval.get(math = repcap.
↪Math.Default)
```

Sets the y-axis values for calculating the reference curve in power measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
yval: float Range: -200 to 100

**set**(*yval: float*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:POWer:MATH<CH>:YVAL
driver.calculate.power.sweep.power.math.yval.set(yval = 1.0, math = repcap.Math.
↪Default)
```

Sets the y-axis values for calculating the reference curve in power measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param yval**
float Range: -200 to 100

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.3 Time

**class TimeCls**

Time commands group definition. 12 total commands, 3 Subgroups, 0 group commands

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.time.clone()
```

##### Subgroups

###### 6.1.1.1.3.1 Gate<Gate>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.calculate.power.sweep.time.gate.repcap_gate_get()
driver.calculate.power.sweep.time.gate.repcap_gate_set(repcap.Gate.Nr1)
```

**class GateCls**

Gate commands group definition. 6 total commands, 6 Subgroups, 0 group commands Repeated Capability: Gate, default value after init: Gate.Nr1

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.time.gate.clone()
```

##### Subgroups

###### 6.1.1.1.3.2 Average

##### SCPI Command :

```
CALCulate:[POWer]:SWEep:TIME:GATE<CH>:AVERage
```

**class AverageCls**

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

**get**(*gate=Gate.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:AVERage
value: float = driver.calculate.power.sweep.time.gate.average.get(gate = repcap.
↪Gate.Default)
```

Queries the average power value of the time gated measurement.

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

**return**
average: float Range: -1000 to 1000

#### 6.1.1.1.3.3 Feed

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:TIME:GATE<CH>:FEED
```

**class FeedCls**

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

**get**(*gate=Gate.Default*) → MeasRespTimeGate

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:FEED
value: enums.MeasRespTimeGate = driver.calculate.power.sweep.time.gate.feed.
↪get(gate = repcap.Gate.Default)
```

Selects the trace for time gated measurement. Both gates are assigned to the same trace.

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

**return**
feed: TRAC1| TRAC2| TRAC3| TRACe1| TRACe2| TRACe3| TRAC4| TRACe4

**set**(*feed: MeasRespTimeGate*, *gate=Gate.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:FEED
driver.calculate.power.sweep.time.gate.feed.set(feed = enums.MeasRespTimeGate.
↪TRAC1, gate = repcap.Gate.Default)
```

Selects the trace for time gated measurement. Both gates are assigned to the same trace.

**param feed**
TRAC1| TRAC2| TRAC3| TRACe1| TRACe2| TRACe3| TRAC4| TRACe4

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

#### 6.1.1.1.3.4 Maximum

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:TIME:GATE<CH>:MAXimum
```

**class MaximumCls**

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

**get**(*gate=Gate.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:MAXimum
value: float = driver.calculate.power.sweep.time.gate.maximum.get(gate = repcap.
↪Gate.Default)
```

Queries the average power value of the time gated measurement.

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

**return**
maximum: float Range: -1000 to 1000

#### 6.1.1.1.3.5 Start

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STARt
```

**class StartCls**

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

**get**(*gate=Gate.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STARt
value: float = driver.calculate.power.sweep.time.gate.start.get(gate = repcap.
↪Gate.Default)
```

Sets the start time of the selected gate. Insert value and unit.

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

**return**
start: No help available

**set**(*start: float*, *gate=Gate.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STARt
driver.calculate.power.sweep.time.gate.start.set(start = 1.0, gate = repcap.
↪Gate.Default)
```

Sets the start time of the selected gate. Insert value and unit.

**param start**
float

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

#### 6.1.1.1.3.6 State

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STATe
```

**class StateCls**

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

**get**(*gate=Gate.Default*) → bool

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STATe
value: bool = driver.calculate.power.sweep.time.gate.state.get(gate = repcap.
↪Gate.Default)
```

Activates the gate settings for the selected trace. The measurement is started with command SENS:POW:INIT. Both gates are active at one time.

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *gate=Gate.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STATe
driver.calculate.power.sweep.time.gate.state.set(state = False, gate = repcap.
↪Gate.Default)
```

Activates the gate settings for the selected trace. The measurement is started with command SENS:POW:INIT. Both gates are active at one time.

**param state**
0| 1| OFF| ON

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

#### 6.1.1.1.3.7 Stop

##### SCPI Command :

```
CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STOP
```

**class StopCls**

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

**get**(*gate=Gate.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STOP
value: float = driver.calculate.power.sweep.time.gate.stop.get(gate = repcap.
↪Gate.Default)
```

Sets the start time of the selected gate. Insert value and unit.

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

**return**
stop: float

**set**(*stop: float*, *gate=Gate.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:GATE<CH>:STOP
driver.calculate.power.sweep.time.gate.stop.set(stop = 1.0, gate = repcap.Gate.
↪Default)
```

Sets the start time of the selected gate. Insert value and unit.

**param stop**
float

**param gate**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Gate')

#### 6.1.1.1.3.8 Marker<Marker>

##### RepCap Settings

```
# Range: Nr0 .. Nr31
rc = driver.calculate.power.sweep.time.marker.repcap_marker_get()
driver.calculate.power.sweep.time.marker.repcap_marker_set(repcap.Marker.Nr0)
```

**class MarkerCls**

Marker commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Marker, default value after init: Marker.Nr0

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.time.marker.clone()
```

### Subgroups

#### 6.1.1.1.3.9 Feed

### SCPI Command :

```
CALCulate:[POWer]:SWEep:TIME:MARKer<CH>:FEED
```

**class FeedCls**

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

**get**(*marker=Marker.Default*) → MeasRespTimeGate

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MARKer<CH>:FEED
value: enums.MeasRespTimeGate = driver.calculate.power.sweep.time.marker.feed.
↪get(marker = repcap.Marker.Default)
```

No command help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
marker_binding: No help available

**set**(*marker_binding: MeasRespTimeGate*, *marker=Marker.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MARKer<CH>:FEED
driver.calculate.power.sweep.time.marker.feed.set(marker_binding = enums.
↪MeasRespTimeGate.TRAC1, marker = repcap.Marker.Default)
```

No command help available

**param marker_binding**
No help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.1.1.1.3.10 State

### SCPI Command :

```
CALCulate:[POWer]:SWEep:TIME:MARKer<CH>:STATe
```

**class StateCls**

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

**get**(*marker=Marker.Default*) → bool

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MARKer<CH>:STATe
value: bool = driver.calculate.power.sweep.time.marker.state.get(marker =␣
↪repcap.Marker.Default)
```

No command help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
marker_state: No help available

**set**(*marker_state: bool*, *marker=Marker.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MARKer<CH>:STATe
driver.calculate.power.sweep.time.marker.state.set(marker_state = False, marker␣
↪= repcap.Marker.Default)
```

No command help available

**param marker_state**
No help available

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.1.1.1.3.11 Math<Math>

### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.calculate.power.sweep.time.math.repcap_math_get()
driver.calculate.power.sweep.time.math.repcap_math_set(repcap.Math.Nr1)
```

**class MathCls**

Math commands group definition. 4 total commands, 4 Subgroups, 0 group commands Repeated Capability: Math, default value after init: Math.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calculate.power.sweep.time.math.clone()
```

### Subgroups

#### 6.1.1.1.3.12 State

### SCPI Command :

```
CALCulate:[POWer]:SWEep:TIME:MATH<CH>:STATe
```

**class StateCls**

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

**get**(*math=Math.Default*) → bool

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:STATe
value: bool = driver.calculate.power.sweep.time.math.state.get(math = repcap.
↪Math.Default)
```

Activates the trace mathematics mode for 'Time' measurement. This feature enables you to calculate the difference between the measurement values of two traces. For further calculation, a math result can also be assigned to a trace.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:STATe
driver.calculate.power.sweep.time.math.state.set(state = False, math = repcap.
↪Math.Default)
```

Activates the trace mathematics mode for 'Time' measurement. This feature enables you to calculate the difference between the measurement values of two traces. For further calculation, a math result can also be assigned to a trace.

**param state**
0| 1| OFF| ON

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.3.13 Subtract

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:TIME:MATH<CH>:SUBTract
```

**class SubtractCls**

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

**get**(*math=Math.Default*) → MeasRespMath

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:SUBTract
value: enums.MeasRespMath = driver.calculate.power.sweep.time.math.subtract.
↪get(math = repcap.Math.Default)
```

Subtracts the operands 1 and 2 and assigns the result to the selected trace in 'Time' measurement mode.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
subtract: T1T1| T1T2| T1T3| T1T4| T1REf| T2T1| T2T2| T2T3| T2T4| T2REf| T3T1| T3T2| T3T3| T3T4| T3REf| T4T1| T4T2| T4T3| T4T4| T4REf

**set**(*subtract: MeasRespMath*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:SUBTract
driver.calculate.power.sweep.time.math.subtract.set(subtract = enums.
↪MeasRespMath.T1REf, math = repcap.Math.Default)
```

Subtracts the operands 1 and 2 and assigns the result to the selected trace in 'Time' measurement mode.

**param subtract**
T1T1| T1T2| T1T3| T1T4| T1REf| T2T1| T2T2| T2T3| T2T4| T2REf| T3T1| T3T2| T3T3| T3T4| T3REf| T4T1| T4T2| T4T3| T4T4| T4REf

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.3.14 Xval

**SCPI Command :**

```
CALCulate:[POWer]:SWEep:TIME:MATH<CH>:XVAL
```

**class XvalCls**

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

**get**(*math=Math.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:XVAL
value: float = driver.calculate.power.sweep.time.math.xval.get(math = repcap.
↪Math.Default)
```

Sets the x-axis values for calculating the reference curve in time measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
xval: float Range: 0 to 2

**set**(*xval: float*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:XVAL
driver.calculate.power.sweep.time.math.xval.set(xval = 1.0, math = repcap.Math.
↪Default)
```

Sets the x-axis values for calculating the reference curve in time measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param xval**
float Range: 0 to 2

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

#### 6.1.1.1.3.15 Yval

##### SCPI Command :

```
CALCulate:[POWer]:SWEep:TIME:MATH<CH>:YVAL
```

**class YvalCls**

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

**get**(*math=Math.Default*) → float

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:YVAL
value: float = driver.calculate.power.sweep.time.math.yval.get(math = repcap.
↪Math.Default)
```

Sets the y-axis values for calculating the reference curve in time measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

**return**
yval: float Range: -200 to 100

**set**(*yval: float*, *math=Math.Default*) → None

```
# SCPI: CALCulate:[POWer]:SWEep:TIME:MATH<CH>:YVAL
driver.calculate.power.sweep.time.math.yval.set(yval = 1.0, math = repcap.Math.
↪Default)
```

Sets the y-axis values for calculating the reference curve in time measurement mode. To determine two points ('Point A'/'Point B') , set suffix 1 and 2 in keyword MATH<ch>.

**param yval**
float Range: -200 to 100

**param math**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Math')

## 6.2 Calibration

### SCPI Commands :

```
CALibration<HW>:CONTinueonerror
CALibration<HW>:DEBug
```

**class CalibrationCls**

Calibration commands group definition. 44 total commands, 13 Subgroups, 2 group commands

**get_continue_on_error**() → bool

```
# SCPI: CALibration<HW>:CONTinueonerror
value: bool = driver.calibration.get_continue_on_error()
```

Continues the calibration even though an error was detected. By default adjustments are aborted on error.

**return**
state: 1| ON| 0| OFF

**set_continue_on_error**(*state: bool*) → None

```
# SCPI: CALibration<HW>:CONTinueonerror
driver.calibration.set_continue_on_error(state = False)
```

Continues the calibration even though an error was detected. By default adjustments are aborted on error.

**param state**
1| ON| 0| OFF

**set_debug**(*state: bool*) → None

```
# SCPI: CALibration<HW>:DEBug
driver.calibration.set_debug(state = False)
```

Activates logging of the internal adjustments.

**param state**
0| 1| OFF| ON

#### Cloning the Group

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

#### Subgroups

### 6.2.1 All

**class AllCls**

All 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.calibration.all.clone()
```

#### Subgroups

##### 6.2.1.1 Measure

#### SCPI Command :

```
CALibration:ALL:[MEASure]
```

**class MeasureCls**

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

**get**(*force: str = None*) → bool

```
# SCPI: CALibration:ALL:[MEASure]
value: bool = driver.calibration.all.measure.get(force = 'abc')
```

Starts all internal adjustments that do not need external measuring equipment. NOTICE! High power at the RF output applied during internal adjustment can destroy a connected DUT (device under test) . How to: See 'Running internal adjustments'.

**param force**
string

**return**
measure: 1| ON| 0| OFF

### 6.2.2 Csynthesis

#### SCPI Command :

```
CALibration:CSYNthesis:[MEASure]
```

**class CsynthesisCls**

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

**get_measure**() → bool

```
# SCPI: CALibration:CSYNthesis:[MEASure]
value: bool = driver.calibration.csynthesis.get_measure()
```

No command help available

**return**
measure: No help available

### 6.2.3 Data

#### SCPI Command :

```
CALibration:DATA:EXPort
```

**class DataCls**

Data commands group definition. 5 total commands, 3 Subgroups, 1 group commands

**export**() → None

```
# SCPI: CALibration:DATA:EXPort
driver.calibration.data.export()
```

No command help available

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

```
# SCPI: CALibration:DATA:EXPort
driver.calibration.data.export_with_opc()
```

No command help available

Same as export, but waits for the operation to complete before continuing further. Use the RsSmab.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.calibration.data.clone()
```

#### Subgroups

##### 6.2.3.1 Factory

#### SCPI Command :

```
CALibration:DATA:FACTory:DATE
```

**class FactoryCls**

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

**get_date**() → str

```
# SCPI: CALibration:DATA:FACTory:DATE
value: str = driver.calibration.data.factory.get_date()
```

Queries the date of the last factory calibration.

**return**
date: string

##### 6.2.3.2 Remove

#### SCPI Command :

```
CALibration:DATA:REMove
```

**class RemoveCls**

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

**set**() → None

```
# SCPI: CALibration:DATA:REMove
driver.calibration.data.remove.set()
```

No command help available

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

```
# SCPI: CALibration:DATA:REMove
driver.calibration.data.remove.set_with_opc()
```

No command help available

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

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

#### 6.2.3.3 Update

**SCPI Command :**

```
CALibration<HW>:DATA:UPDate
```

**class UpdateCls**

Update commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**set_value**(*action_sel: CalDataUpdate*) → None

```
# SCPI: CALibration<HW>:DATA:UPDate
driver.calibration.data.update.set_value(action_sel = enums.CalDataUpdate.BBFRC)
```

No command help available

**param action_sel**
No help available

**Cloning the Group**

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

**Subgroups**

##### 6.2.3.3.1 Level

**class LevelCls**

Level 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.calibration.data.update.level.clone()
```

**Subgroups**

###### 6.2.3.3.1.1 Force

**SCPI Command :**

```
CALibration<HW>:DATA:UPDate:LEVel:FORCe
```

**class ForceCls**

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

**set**() → None

```
# SCPI: CALibration<HW>:DATA:UPDate:LEVel:FORCe
driver.calibration.data.update.level.force.set()
```

No command help available

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

```
# SCPI: CALibration<HW>:DATA:UPDate:LEVel:FORCe
driver.calibration.data.update.level.force.set_with_opc()
```

No command help available

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

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

### 6.2.4 Delay

#### SCPI Commands :

```
CALibration:DELay:MINutes
CALibration:DELay:[MEASure]
```

**class DelayCls**

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

**get_measure**() → bool

```
# SCPI: CALibration:DELay:[MEASure]
value: bool = driver.calibration.delay.get_measure()
```

Starts the delayed adjustment process. When the warm-up time has elapsed (see method RsSmab.Calibration.Delay.minutes, it executes the internal adjustments. If you have enabled automatic shutdown, CALibration:DELay:SHUTdown[:STATe] ON, the instrument shuts down when the adjustments are completed.

**return**
error: 1| ON| 0| OFF

**get_minutes**() → int

```
# SCPI: CALibration:DELay:MINutes
value: int = driver.calibration.delay.get_minutes()
```

Sets the warm-up time to wait before internal adjustment starts automatically. Automatic execution starts only, if you have enabled the calibration with command ON.

**return**
minutes: integer Range: 30 to 120

**set_minutes**(*minutes: int*) → None

```
# SCPI: CALibration:DELay:MINutes
driver.calibration.delay.set_minutes(minutes = 1)
```

Sets the warm-up time to wait before internal adjustment starts automatically. Automatic execution starts only, if you have enabled the calibration with command ON.

**param minutes**
integer Range: 30 to 120

### Cloning the Group

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

### Subgroups

#### 6.2.4.1 Shutdown

### SCPI Command :

```
CALibration:DELay:SHUTdown:[STATe]
```

**class ShutdownCls**

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

**get_state**() → bool

```
# SCPI: CALibration:DELay:SHUTdown:[STATe]
value: bool = driver.calibration.delay.shutdown.get_state()
```

Enables the instrument to shut down automatically after calibration.

**return**
shutdown: 1| ON| 0| OFF

**set_state**(*shutdown: bool*) → None

```
# SCPI: CALibration:DELay:SHUTdown:[STATe]
driver.calibration.delay.shutdown.set_state(shutdown = False)
```

Enables the instrument to shut down automatically after calibration.

**param shutdown**
1| ON| 0| OFF

### 6.2.5 Detector

**class DetectorCls**

Detector 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.calibration.detector.clone()
```

#### Subgroups

##### 6.2.5.1 RfLevel

#### SCPI Commands :

```
CALibration:DETector:RFLevel:EXPected
CALibration:DETector:RFLevel
```

**class RfLevelCls**

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

**get_expected**() → float

```
# SCPI: CALibration:DETector:RFLevel:EXPected
value: float = driver.calibration.detector.rfLevel.get_expected()
```

No command help available

**return**
level_value_exp: No help available

**get_value**() → float

```
# SCPI: CALibration:DETector:RFLevel
value: float = driver.calibration.detector.rfLevel.get_value()
```

No command help available

**return**
level_value: No help available

### 6.2.6 FmOffset

#### SCPI Command :

```
CALibration<HW>:FMOFfset:[MEASure]
```

**class FmOffsetCls**

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

**get_measure**() → bool

```
# SCPI: CALibration<HW>:FMOFfset:[MEASure]
value: bool = driver.calibration.fmOffset.get_measure()
```

No command help available

**return**
measure: No help available

### 6.2.7 Frequency

#### SCPI Commands :

```
CALibration:FREQuency:SWPoints
CALibration<HW>:FREQuency:[MEASure]
```

**class FrequencyCls**

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

**get_measure**() → bool

```
# SCPI: CALibration<HW>:FREQuency:[MEASure]
value: bool = driver.calibration.frequency.get_measure()
```

No command help available

**return**
measure: No help available

**get_sw_points**() → str

```
# SCPI: CALibration:FREQuency:SWPoints
value: str = driver.calibration.frequency.get_sw_points()
```

No command help available

**return**
freq_switch_point: No help available

**set_sw_points**(*freq_switch_point: str*) → None

```
# SCPI: CALibration:FREQuency:SWPoints
driver.calibration.frequency.set_sw_points(freq_switch_point = 'abc')
```

No command help available

**param freq_switch_point**
No help available

### 6.2.8 Level

#### SCPI Commands :

```
CALibration:LEVel:BWIDth
CALibration<HW>:LEVel:STATe
```

**class LevelCls**

Level commands group definition. 17 total commands, 8 Subgroups, 2 group commands

**get_bandwidth**() → CalPowBandwidth

```
# SCPI: CALibration:LEVel:BWIDth
value: enums.CalPowBandwidth = driver.calibration.level.get_bandwidth()
```

No command help available

**return**
bandwidth: No help available

**get_state**() → StateExtended

```
# SCPI: CALibration<HW>:LEVel:STATe
value: enums.StateExtended = driver.calibration.level.get_state()
```

No command help available

**return**
state: No help available

**set_bandwidth**(*bandwidth: CalPowBandwidth*) → None

```
# SCPI: CALibration:LEVel:BWIDth
driver.calibration.level.set_bandwidth(bandwidth = enums.CalPowBandwidth.AUTO)
```

No command help available

**param bandwidth**
No help available

**set_state**(*state: StateExtended*) → None

```
# SCPI: CALibration<HW>:LEVel:STATe
driver.calibration.level.set_state(state = enums.StateExtended._0)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.2.8.1 Alinearize

### SCPI Command :

```
CALibration:LEVel:ALINearize:MODE
```

**class AlinearizeCls**

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

**get_mode**() → CalPowActorLinMode

```
# SCPI: CALibration:LEVel:ALINearize:MODE
value: enums.CalPowActorLinMode = driver.calibration.level.alinearize.get_mode()
```

No command help available

**return**
mode: No help available

**set_mode**(*mode: CalPowActorLinMode*) → None

```
# SCPI: CALibration:LEVel:ALINearize:MODE
driver.calibration.level.alinearize.set_mode(mode = enums.CalPowActorLinMode.
↪AUTO)
```

No command help available

**param mode**
No help available

#### 6.2.8.2 Amplifier

**class AmplifierCls**

Amplifier 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.calibration.level.amplifier.clone()
```

#### Subgroups

##### 6.2.8.2.1 Stage

#### SCPI Commands :

```
CALibration:LEVel:AMPLifier:STAGe:FREQuenz
CALibration:LEVel:AMPLifier:STAGe:MODE
CALibration:LEVel:AMPLifier:STAGe:SUB
CALibration:LEVel:AMPLifier:STAGe
```

**class StageCls**

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

**get_frequency**() → int

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe:FREQuenz
value: int = driver.calibration.level.amplifier.stage.get_frequency()
```

No command help available

**return**
freq_stage: No help available

**get_mode**() → StagMode

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe:MODE
value: enums.StagMode = driver.calibration.level.amplifier.stage.get_mode()
```

No command help available

**return**
mode: No help available

**get_sub**() → int

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe:SUB
value: int = driver.calibration.level.amplifier.stage.get_sub()
```

No command help available

**return**
sub_stage: No help available

**get_value**() → int

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe
value: int = driver.calibration.level.amplifier.stage.get_value()
```

No command help available

**return**
stage: No help available

**set_frequency**(*freq_stage: int*) → None

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe:FREQuenz
driver.calibration.level.amplifier.stage.set_frequency(freq_stage = 1)
```

No command help available

**param freq_stage**
No help available

`set_mode`(*mode: StagMode*) → None

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe:MODE
driver.calibration.level.amplifier.stage.set_mode(mode = enums.StagMode.AUTO)
```

No command help available

**param mode**
No help available

`set_sub`(*sub_stage: int*) → None

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe:SUB
driver.calibration.level.amplifier.stage.set_sub(sub_stage = 1)
```

No command help available

**param sub_stage**
No help available

`set_value`(*stage: int*) → None

```
# SCPI: CALibration:LEVel:AMPLifier:STAGe
driver.calibration.level.amplifier.stage.set_value(stage = 1)
```

No command help available

**param stage**
No help available

#### 6.2.8.3 Attenuator

**SCPI Commands :**

```
CALibration<HW>:LEVel:ATTenuator:MODE
CALibration<HW>:LEVel:ATTenuator:STAGe
```

`class AttenuatorCls`

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

`get_mode`() → CalPowAttMode

```
# SCPI: CALibration<HW>:LEVel:ATTenuator:MODE
value: enums.CalPowAttMode = driver.calibration.level.attenuator.get_mode()
```

No command help available

**return**
mode: No help available

**get_stage**() → int

```
# SCPI: CALibration<HW>:LEVel:ATTenuator:STAGe
value: int = driver.calibration.level.attenuator.get_stage()
```

No command help available

**return**
stage: No help available

**set_stage**(*stage: int*) → None

```
# SCPI: CALibration<HW>:LEVel:ATTenuator:STAGe
driver.calibration.level.attenuator.set_stage(stage = 1)
```

No command help available

**param stage**
No help available

#### 6.2.8.4 DetAtt

**SCPI Command :**

```
CALibration:LEVel:DETatt:MODE
```

**class DetAttCls**

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

**get_mode**() → CalPowAmpDetMode

```
# SCPI: CALibration:LEVel:DETatt:MODE
value: enums.CalPowAmpDetMode = driver.calibration.level.detAtt.get_mode()
```

No command help available

**return**
mode: No help available

**set_mode**(*mode: CalPowAmpDetMode*) → None

```
# SCPI: CALibration:LEVel:DETatt:MODE
driver.calibration.level.detAtt.set_mode(mode = enums.CalPowAmpDetMode.AMP)
```

No command help available

**param mode**
No help available

#### 6.2.8.5 Dlinearize

**SCPI Command :**

```
CALibration:LEVel:DLINearize:MODE
```

**class DlinearizeCls**

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

**get_mode**() → CalPowDetLinMode

```
# SCPI: CALibration:LEVel:DLINearize:MODE
value: enums.CalPowDetLinMode = driver.calibration.level.dlinearize.get_mode()
```

No command help available

**return**
mode: No help available

**set_mode**(*mode: CalPowDetLinMode*) → None

```
# SCPI: CALibration:LEVel:DLINearize:MODE
driver.calibration.level.dlinearize.set_mode(mode = enums.CalPowDetLinMode.AUTO)
```

No command help available

**param mode**
No help available

#### 6.2.8.6 Measure

**SCPI Command :**

```
CALibration<HW>:LEVel:[MEASure]
```

**class MeasureCls**

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

**get**(*force: str = None*) → bool

```
# SCPI: CALibration<HW>:LEVel:[MEASure]
value: bool = driver.calibration.level.measure.get(force = 'abc')
```

No command help available

**param force**
No help available

**return**
measure: No help available

#### 6.2.8.7 Opu

**class OpuCls**

Opu 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.calibration.level.opu.clone()
```

### Subgroups

#### 6.2.8.7.1 Lcon

### SCPI Command :

```
CALibration:LEVel:OPU:LCON:MODE
```

**class LconCls**

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

**get_mode**() → CalPowOpuLconMode

```
# SCPI: CALibration:LEVel:OPU:LCON:MODE
value: enums.CalPowOpuLconMode = driver.calibration.level.opu.lcon.get_mode()
```

No command help available

**return**
lcon_mode: No help available

**set_mode**(*lcon_mode: CalPowOpuLconMode*) → None

```
# SCPI: CALibration:LEVel:OPU:LCON:MODE
driver.calibration.level.opu.lcon.set_mode(lcon_mode = enums.CalPowOpuLconMode.
↪AM)
```

No command help available

**param lcon_mode**
No help available

#### 6.2.8.7.2 Stage

### SCPI Commands :

```
CALibration:LEVel:OPU:STAGe:MODE
CALibration:LEVel:OPU:STAGe:SUB
CALibration:LEVel:OPU:STAGe
```

class StageCls

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

get_mode() → StagMode

```
# SCPI: CALibration:LEVel:OPU:STAGe:MODE
value: enums.StagMode = driver.calibration.level.opu.stage.get_mode()
```

No command help available

**return**
stage_mode: No help available

get_sub() → int

```
# SCPI: CALibration:LEVel:OPU:STAGe:SUB
value: int = driver.calibration.level.opu.stage.get_sub()
```

No command help available

**return**
stage_sub: No help available

get_value() → int

```
# SCPI: CALibration:LEVel:OPU:STAGe
value: int = driver.calibration.level.opu.stage.get_value()
```

No command help available

**return**
stage: No help available

set_mode(*stage_mode: StagMode*) → None

```
# SCPI: CALibration:LEVel:OPU:STAGe:MODE
driver.calibration.level.opu.stage.set_mode(stage_mode = enums.StagMode.AUTO)
```

No command help available

**param stage_mode**
No help available

set_sub(*stage_sub: int*) → None

```
# SCPI: CALibration:LEVel:OPU:STAGe:SUB
driver.calibration.level.opu.stage.set_sub(stage_sub = 1)
```

No command help available

**param stage_sub**
No help available

set_value(*stage: int*) → None

```
# SCPI: CALibration:LEVel:OPU:STAGe
driver.calibration.level.opu.stage.set_value(stage = 1)
```

No command help available

**param stage**
No help available

#### 6.2.8.8 SwAmplifier

**SCPI Command :**

```
CALibration:LEVel:SWAMplifier:STATe
```

**class SwAmplifierCls**

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

**get_state**() → bool

```
# SCPI: CALibration:LEVel:SWAMplifier:STATe
value: bool = driver.calibration.level.swAmplifier.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: CALibration:LEVel:SWAMplifier:STATe
driver.calibration.level.swAmplifier.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.2.9 LfOutput

**SCPI Command :**

```
CALibration:LFOutput:[MEASure]
```

**class LfOutputCls**

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

**get_measure**() → bool

```
# SCPI: CALibration:LFOutput:[MEASure]
value: bool = driver.calibration.lfOutput.get_measure()
```

No command help available

**return**
measure: No help available

### 6.2.10 Mode

#### SCPI Commands :

```
CALibration:MODE:CONFiguration
CALibration:MODE
```

**class ModeCls**

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

**get_configuration**() → str

```
# SCPI: CALibration:MODE:CONFiguration
value: str = driver.calibration.mode.get_configuration()
```

No command help available

**return**
cal_conf_xml: No help available

**get_value**() → CalAdjMode

```
# SCPI: CALibration:MODE
value: enums.CalAdjMode = driver.calibration.mode.get_value()
```

No command help available

**return**
cal_mode: No help available

**set_configuration**(*cal_conf_xml: str*) → None

```
# SCPI: CALibration:MODE:CONFiguration
driver.calibration.mode.set_configuration(cal_conf_xml = 'abc')
```

No command help available

**param cal_conf_xml**
No help available

**set_value**(*cal_mode: CalAdjMode*) → None

```
# SCPI: CALibration:MODE
driver.calibration.mode.set_value(cal_mode = enums.CalAdjMode.BURNin)
```

No command help available

**param cal_mode**
No help available

### 6.2.11 Roscillator

**class RoscillatorCls**

Roscillator commands group definition. 3 total commands, 2 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.2.11.1 Data

#### SCPI Commands :

```
CALibration:ROSCillator:DATA:MODE
CALibration:ROSCillator:[DATA]
```

**class DataCls**

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

**get_mode**() → CalDataMode

```
# SCPI: CALibration:ROSCillator:DATA:MODE
value: enums.CalDataMode = driver.calibration.roscillator.data.get_mode()
```

No command help available

**return**
mode: No help available

**get_value**() → int

```
# SCPI: CALibration:ROSCillator:[DATA]
value: int = driver.calibration.roscillator.data.get_value()
```

No command help available

**return**
data: No help available

**set_mode**(*mode: CalDataMode*) → None

```
# SCPI: CALibration:ROSCillator:DATA:MODE
driver.calibration.roscillator.data.set_mode(mode = enums.CalDataMode.CUSTomer)
```

No command help available

**param mode**
No help available

**set_value**(*data: int*) → None

```
# SCPI: CALibration:ROSCillator:[DATA]
driver.calibration.roscillator.data.set_value(data = 1)
```

No command help available

**param data**
No help available

#### 6.2.11.2 Store

**SCPI Command :**

```
CALibration:ROSCillator:STORe
```

**class StoreCls**

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

**set**() → None

```
# SCPI: CALibration:ROSCillator:STORe
driver.calibration.roscillator.store.set()
```

No command help available

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

```
# SCPI: CALibration:ROSCillator:STORe
driver.calibration.roscillator.store.set_with_opc()
```

No command help available

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

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

### 6.2.12 Selected

**class SelectedCls**

Selected 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.calibration.selected.clone()
```

#### Subgroups

##### 6.2.12.1 Measure

#### SCPI Command :

```
CALibration:SELected:[MEASure]
```

**class MeasureCls**

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

**get**(*to_test_args: str*) → TestCalSelected

```
# SCPI: CALibration:SELected:[MEASure]
value: enums.TestCalSelected = driver.calibration.selected.measure.get(to_test_
↪args = 'abc')
```

No command help available

**param to_test_args**
No help available

**return**
test_result: No help available

### 6.2.13 Tselected

#### SCPI Commands :

```
CALibration:TSELected:CATalog
CALibration:TSELected:STEP
CALibration:TSELected:[MEASure]
```

**class TselectedCls**

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

**get_catalog**() → str

```
# SCPI: CALibration:TSELected:CATalog
value: str = driver.calibration.tselected.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_measure**() → bool

```
# SCPI: CALibration:TSELected:[MEASure]
value: bool = driver.calibration.tselected.get_measure()
```

No command help available

**return**
meas: No help available

**get_step**() → str

```
# SCPI: CALibration:TSELected:STEP
value: str = driver.calibration.tselected.get_step()
```

No command help available

**return**
sel_string: No help available

**set_step**(*sel_string: str*) → None

```
# SCPI: CALibration:TSELected:STEP
driver.calibration.tselected.set_step(sel_string = 'abc')
```

No command help available

**param sel_string**
No help available

## 6.3 Csynthesis

### SCPI Commands :

```
CSYNthesis:OTYPe
CSYNthesis:STATe
CSYNthesis:VOLTage
```

**class CsynthesisCls**

Csynthesis commands group definition. 13 total commands, 4 Subgroups, 3 group commands

**get_otype**() → ClkSynOutType

```
# SCPI: CSYNthesis:OTYPe
value: enums.ClkSynOutType = driver.csynthesis.get_otype()
```

Defines the shape of the generated clock signal.

**return**
mode: SESine| DSQuare| CMOS| DSINe SESine = single-ended sine DSINe = differential sine DSQuare = differential square CMOS = CMOS

**get_state**() → bool

```
# SCPI: CSYNthesis:STATe
value: bool = driver.csynthesis.get_state()
```

Activates the clock synthesis.

**return**
state: 1| ON| 0| OFF

**get_voltage**() → float

```
# SCPI: CSYNthesis:VOLTage
value: float = driver.csynthesis.get_voltage()
```

Sets the voltage for the CMOS signal.

**return**
voltage: float Range: 0.8 to 2.7

**set_otype**(*mode: ClkSynOutType*) → None

```
# SCPI: CSYNthesis:OTYPe
driver.csynthesis.set_otype(mode = enums.ClkSynOutType.CMOS)
```

Defines the shape of the generated clock signal.

**param mode**
SESine| DSQuare| CMOS| DSINe SESine = single-ended sine DSINe = differential sine DSQuare = differential square CMOS = CMOS

**set_state**(*state: bool*) → None

```
# SCPI: CSYNthesis:STATe
driver.csynthesis.set_state(state = False)
```

Activates the clock synthesis.

**param state**
1| ON| 0| OFF

**set_voltage**(*voltage: float*) → None

```
# SCPI: CSYNthesis:VOLTage
driver.csynthesis.set_voltage(voltage = 1.0)
```

Sets the voltage for the CMOS signal.

**param voltage**
float Range: 0.8 to 2.7

#### Cloning the Group

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

#### Subgroups

### 6.3.1 Frequency

#### SCPI Command :

```
CSYNthesis:FREQuency
```

**class FrequencyCls**

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

**get_value**() → float

```
# SCPI: CSYNthesis:FREQuency
value: float = driver.csynthesis.frequency.get_value()
```

Sets the frequency of the generated clock signal.

**return**
frequency: float Numerical value Sets the frequency UP|DOWN Varies the frequency step by step. The frequency is increased or decreased by the value set with the command method RsSmab.Csynthesis.Frequency.Step.value. Range: 100E3 to 1.5E9

**set_value**(*frequency: float*) → None

```
# SCPI: CSYNthesis:FREQuency
driver.csynthesis.frequency.set_value(frequency = 1.0)
```

Sets the frequency of the generated clock signal.

**param frequency**
float Numerical value Sets the frequency UP|DOWN Varies the frequency step by step. The frequency is increased or decreased by the value set with the command method RsSmab.Csynthesis.Frequency.Step.value. Range: 100E3 to 1.5E9

#### Cloning the Group

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

### Subgroups

#### 6.3.1.1 Step

### SCPI Commands :

```
CSYNthesis:FREQuency:STEP:MODE
CSYNthesis:FREQuency:STEP
```

**class StepCls**

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

**get_mode**() → FreqStepMode

```
# SCPI: CSYNthesis:FREQuency:STEP:MODE
value: enums.FreqStepMode = driver.csynthesis.frequency.step.get_mode()
```

Defines the type of step size to vary the frequency and level at discrete steps.

**return**
mode: DECimal| USER DECimal Increases or decreases the level in steps of 10. USER Increases or decreases the value in increments, set with the command: method RsSmab.Csynthesis.Frequency.Step.value method RsSmab.Csynthesis.Power.Step.increment

**get_value**() → float

```
# SCPI: CSYNthesis:FREQuency:STEP
value: float = driver.csynthesis.frequency.step.get_value()
```

Sets the step width of the rotary knob and, in user-defined step mode, increases or decreases the frequency.

**return**
step: float Range: 0 to 14999E5

**set_mode**(*mode: FreqStepMode*) → None

```
# SCPI: CSYNthesis:FREQuency:STEP:MODE
driver.csynthesis.frequency.step.set_mode(mode = enums.FreqStepMode.DECimal)
```

Defines the type of step size to vary the frequency and level at discrete steps.

**param mode**
DECimal| USER DECimal Increases or decreases the level in steps of 10. USER Increases or decreases the value in increments, set with the command: method RsSmab.Csynthesis.Frequency.Step.value method RsSmab.Csynthesis.Power.Step.increment

**set_value**(*step: float*) → None

```
# SCPI: CSYNthesis:FREQuency:STEP
driver.csynthesis.frequency.step.set_value(step = 1.0)
```

Sets the step width of the rotary knob and, in user-defined step mode, increases or decreases the frequency.

**param step**
float Range: 0 to 14999E5

### 6.3.2 Offset

#### SCPI Commands :

```
CSYNthesis:OFFSet:STATe
CSYNthesis:OFFSet
```

**class OffsetCls**

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

**get_state**() → bool

```
# SCPI: CSYNthesis:OFFSet:STATe
value: bool = driver.csynthesis.offset.get_state()
```

Activates a DC offset.

**return**
state: 1| ON| 0| OFF

**get_value**() → float

```
# SCPI: CSYNthesis:OFFSet
value: float = driver.csynthesis.offset.get_value()
```

Sets the value of the DC offset.

**return**
offset: float Range: -5 to 5

**set_state**(*state: bool*) → None

```
# SCPI: CSYNthesis:OFFSet:STATe
driver.csynthesis.offset.set_state(state = False)
```

Activates a DC offset.

**param state**
1| ON| 0| OFF

**set_value**(*offset: float*) → None

```
# SCPI: CSYNthesis:OFFSet
driver.csynthesis.offset.set_value(offset = 1.0)
```

Sets the value of the DC offset.

**param offset**
float Range: -5 to 5

### 6.3.3 Phase

#### SCPI Command :

```
CSYNthesis:PHASe
```

**class PhaseCls**

Phase commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_value**() → float

```
# SCPI: CSYNthesis:PHASe
value: float = driver.csynthesis.phase.get_value()
```

Shifts the phase of the generated clock signal.

**return**
phase: float Range: -36000 to 36000

**set_value**(*phase: float*) → None

```
# SCPI: CSYNthesis:PHASe
driver.csynthesis.phase.set_value(phase = 1.0)
```

Shifts the phase of the generated clock signal.

**param phase**
float Range: -36000 to 36000

#### Cloning the Group

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

#### Subgroups

##### 6.3.3.1 Reference

#### SCPI Command :

```
CSYNthesis:PHASe:REFerence
```

**class ReferenceCls**

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

**set**() → None

```
# SCPI: CSYNthesis:PHASe:REFerence
driver.csynthesis.phase.reference.set()
```

Resets the delta phase value.

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

```
# SCPI: CSYNthesis:PHASe:REFerence
driver.csynthesis.phase.reference.set_with_opc()
```

Resets the delta phase value.

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

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

### 6.3.4 Power

#### SCPI Command :

```
CSYNthesis:POWer
```

**class PowerCls**

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

**get_value**() → float

```
# SCPI: CSYNthesis:POWer
value: float = driver.csynthesis.power.get_value()
```

Sets the power level of the generated clock signal.

**return**
power: float Numerical value Sets the level UP|DOWN Varies the level step by step. The level is increased or decreased by the value set with the command method RsSmab.Csynthesis.Power.Step.increment. Range: -24 to 10

**set_value**(*power: float*) → None

```
# SCPI: CSYNthesis:POWer
driver.csynthesis.power.set_value(power = 1.0)
```

Sets the power level of the generated clock signal.

**param power**
float Numerical value Sets the level UP|DOWN Varies the level step by step. The level is increased or decreased by the value set with the command method RsSmab.Csynthesis.Power.Step.increment. Range: -24 to 10

#### Cloning the Group

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

#### Subgroups

##### 6.3.4.1 Step

#### SCPI Commands :

```
CSYNthesis:POWer:STEP:MODE
CSYNthesis:POWer:STEP:[INCRement]
```

**class StepCls**

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

**get_increment**() → float

```
# SCPI: CSYNthesis:POWer:STEP:[INCRement]
value: float = driver.csynthesis.power.step.get_increment()
```

Sets the step width of the rotary knob and, in user-defined step mode, increases or decreases the level.

**return**
increment: float Range: 0 to 35

**get_mode**() → FreqStepMode

```
# SCPI: CSYNthesis:POWer:STEP:MODE
value: enums.FreqStepMode = driver.csynthesis.power.step.get_mode()
```

Defines the type of step size to vary the frequency and level at discrete steps.

**return**
mode: DECimal| USER DECimal Increases or decreases the level in steps of 10. USER Increases or decreases the value in increments, set with the command: method RsSmab.Csynthesis.Frequency.Step.value method RsSmab.Csynthesis.Power.Step.increment

**set_increment**(*increment: float*) → None

```
# SCPI: CSYNthesis:POWer:STEP:[INCRement]
driver.csynthesis.power.step.set_increment(increment = 1.0)
```

Sets the step width of the rotary knob and, in user-defined step mode, increases or decreases the level.

**param increment**
float Range: 0 to 35

**set_mode**(*mode: FreqStepMode*) → None

```
# SCPI: CSYNthesis:POWer:STEP:MODE
driver.csynthesis.power.step.set_mode(mode = enums.FreqStepMode.DECimal)
```

Defines the type of step size to vary the frequency and level at discrete steps.

**param mode**
DECimal| USER DECimal Increases or decreases the level in steps of 10. USER Increases or decreases the value in increments, set with the command: method RsSmab.Csynthesis.Frequency.Step.value method RsSmab.Csynthesis.Power.Step.increment

## 6.4 Device

### SCPI Command :

```
DEVice:PRESet
```

**class DeviceCls**

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

**preset**() → None

```
# SCPI: DEVice:PRESet
driver.device.preset()
```

Presets all parameters which are not related to the signal path, including the LF generator.

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

```
# SCPI: DEVice:PRESet
driver.device.preset_with_opc()
```

Presets all parameters which are not related to the signal path, including the LF generator.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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.device.clone()
```

### Subgroups

### 6.4.1 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.device.settings.clone()
```

#### Subgroups

##### 6.4.1.1 Backup

#### SCPI Command :

```
DEVice:SETTings:BACKup
```

**class BackupCls**

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

**set**() → None

```
# SCPI: DEVice:SETTings:BACKup
driver.device.settings.backup.set()
```

No command help available

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

```
# SCPI: DEVice:SETTings:BACKup
driver.device.settings.backup.set_with_opc()
```

No command help available

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

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

##### 6.4.1.2 Restore

#### SCPI Command :

```
DEVice:SETTings:RESTore
```

**class RestoreCls**

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

**set**() → None

```
# SCPI: DEVice:SETTings:RESTore
driver.device.settings.restore.set()
```

No command help available

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

```
# SCPI: DEVice:SETTings:RESTore
driver.device.settings.restore.set_with_opc()
```

No command help available

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

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

## 6.5 Diagnostic

**class DiagnosticCls**

Diagnostic commands group definition. 21 total commands, 7 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

### 6.5.1 BgInfo

#### SCPI Commands :

```
DIAGnostic<HW>:BGINfo
DIAGnostic<HW>:BGINfo:CATalog
```

**class BgInfoCls**

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

**get**(*board: str = None*) → str

```
# SCPI: DIAGnostic<HW>:BGINfo
value: str = driver.diagnostic.bgInfo.get(board = 'abc')
```

Queries information on the modules available in the instrument, using the variant and revision state.

**param board**
string Module name, as queried with the command method RsSmab.Diagnostic.BgInfo.catalog. To retrieve a complete list of all modules, omit the parameter. The length of the list is variable and depends on the instrument equipment configuration.

**return**
bg_info: Module name Module stock number incl. variant Module revision Module serial number List of comma-separated entries, one entry per module. Each entry for one module consists of four parts that are separated by space characters.

**get_catalog**() → List[str]

```
# SCPI: DIAGnostic<HW>:BGINfo:CATalog
value: List[str] = driver.diagnostic.bgInfo.get_catalog()
```

Queries the names of the assemblies available in the instrument.

**return**
catalog: string List of all assemblies; the values are separated by commas The length of the list is variable and depends on the instrument equipment configuration.

### 6.5.2 Debug

**class DebugCls**

Debug 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.diagnostic.debug.clone()
```

#### Subgroups

##### 6.5.2.1 Page

#### SCPI Commands :

```
DIAGnostic<HW>:DEBug:PAGE
DIAGnostic<HW>:DEBug:PAGE:CATalog
```

**class PageCls**

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

**get_catalog**() → List[str]

```
# SCPI: DIAGnostic<HW>:DEBug:PAGE:CATalog
value: List[str] = driver.diagnostic.debug.page.get_catalog()
```

No command help available

**return**
diag_debug_page_id_cat: No help available

**set**() → None

```
# SCPI: DIAGnostic<HW>:DEBug:PAGE
driver.diagnostic.debug.page.set()
```

No command help available

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

```
# SCPI: DIAGnostic<HW>:DEBug:PAGE
driver.diagnostic.debug.page.set_with_opc()
```

No command help available

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

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

### 6.5.3 Eeprom<Channel>

#### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.diagnostic.eeprom.repcap_channel_get()
driver.diagnostic.eeprom.repcap_channel_set(repcap.Channel.Nr1)
```

#### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:DELete
```

**class EepromCls**

Eeprom commands group definition. 4 total commands, 3 Subgroups, 1 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

**delete**(*channel=Channel.Default*) → None

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:DELete
driver.diagnostic.eeprom.delete(channel = repcap.Channel.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

**delete_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### Cloning the Group

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

#### Subgroups

##### 6.5.3.1 Bidentifier

**class BidentifierCls**

Bidentifier 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.diagnostic.eeprom.bidentifier.clone()
```

#### Subgroups

##### 6.5.3.1.1 Catalog

#### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:BIDentifier:CATalog
```

**class CatalogCls**

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

**get**(*board_id: List[str]*, *channel=Channel.Default*) → List[str]

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:BIDentifier:CATalog
value: List[str] = driver.diagnostic.eeprom.bidentifier.catalog.get(board_id = [
↪'abc1', 'abc2', 'abc3'], channel = repcap.Channel.Default)
```

No command help available

**param board_id**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

**return**
board_id: No help available

#### 6.5.3.2 Customize

#### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:CUSTomize
```

**class CustomizeCls**

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

**set**(*board: str*, *index: int*, *sub_board: int*, *channel=Channel.Default*) → None

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:CUSTomize
driver.diagnostic.eeprom.customize.set(board = 'abc', index = 1, sub_board = 1,␣
↪channel = repcap.Channel.Default)
```

No command help available

**param board**
No help available

**param index**
No help available

**param sub_board**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

#### 6.5.3.3 Data

**class DataCls**

Data 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.diagnostic.eeprom.data.clone()
```

### Subgroups

#### 6.5.3.3.1 Points

### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:DATA:POINts
```

**class PointsCls**

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

**get**(*board: str*, *sub_board: str*, *channel=Channel.Default*) → int

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:DATA:POINts
value: int = driver.diagnostic.eeprom.data.points.get(board = 'abc', sub_board␣
↪= 'abc', channel = repcap.Channel.Default)
```

No command help available

**param board**
No help available

**param sub_board**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

**return**
points: No help available

### 6.5.4 Info

**class InfoCls**

Info commands group definition. 8 total commands, 3 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.5.4.1 Ecount<ErrorCount>

#### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.diagnostic.info.ecount.repcap_errorCount_get()
driver.diagnostic.info.ecount.repcap_errorCount_set(repcap.ErrorCount.Nr1)
```

#### SCPI Command :

```
DIAGnostic:INFO:ECOunt<CH>
```

**class EcountCls**

Ecount commands group definition. 4 total commands, 3 Subgroups, 1 group commands Repeated Capability: ErrorCount, default value after init: ErrorCount.Nr1

**get**(*errorCount=ErrorCount.Default*) → int

```
# SCPI: DIAGnostic:INFO:ECOunt<CH>
value: int = driver.diagnostic.info.ecount.get(errorCount = repcap.ErrorCount.
↪Default)
```

No command help available

**param errorCount**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ecount')

**return**
ecount: No help available

### Cloning the Group

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

### Subgroups

#### 6.5.4.1.1 Info

### SCPI Command :

```
DIAGnostic:INFO:ECOunt<CH>:INFO
```

**class InfoCls**

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

**get**(*errorCount=ErrorCount.Default*) → str

```
# SCPI: DIAGnostic:INFO:ECOunt<CH>:INFO
value: str = driver.diagnostic.info.ecount.info.get(errorCount = repcap.
↪ErrorCount.Default)
```

No command help available

**param errorCount**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ecount')

**return**
ecount: No help available

#### 6.5.4.1.2 Name

### SCPI Command :

```
DIAGnostic:INFO:ECOunt<CH>:NAME
```

**class NameCls**

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

**get**(*errorCount=ErrorCount.Default*) → str

```
# SCPI: DIAGnostic:INFO:ECOunt<CH>:NAME
value: str = driver.diagnostic.info.ecount.name.get(errorCount = repcap.
↪ErrorCount.Default)
```

No command help available

**param errorCount**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ecount')

**return**
ecount: No help available

#### 6.5.4.1.3 Set

**SCPI Command :**

```
DIAGnostic:INFO:ECOunt<CH>:SET
```

**class SetCls**

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

**set**(*ecount: int*, *errorCount=ErrorCount.Default*) → None

```
# SCPI: DIAGnostic:INFO:ECOunt<CH>:SET
driver.diagnostic.info.ecount.set.set(ecount = 1, errorCount = repcap.
↪ErrorCount.Default)
```

No command help available

**param ecount**
No help available

**param errorCount**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ecount')

### 6.5.4.2 Otime

**SCPI Commands :**

```
DIAGnostic:INFO:OTIMe:SET
DIAGnostic:INFO:OTIMe
```

**class OtimeCls**

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

**get_set**() → int

```
# SCPI: DIAGnostic:INFO:OTIMe:SET
value: int = driver.diagnostic.info.otime.get_set()
```

No command help available

**return**
set_py: No help available

**get_value**() → int

```
# SCPI: DIAGnostic:INFO:OTIMe
value: int = driver.diagnostic.info.otime.get_value()
```

Queries the operating hours of the instrument so far.

**return**
operation_time: integer Range: 0 to INT_MAX

**set_set**(*set_py: int*) → None

```
# SCPI: DIAGnostic:INFO:OTIMe:SET
driver.diagnostic.info.otime.set_set(set_py = 1)
```

No command help available

**param set_py**
No help available

#### 6.5.4.3 PoCount

**SCPI Commands :**

```
DIAGnostic:INFO:POCount:SET
DIAGnostic:INFO:POCount
```

**class PoCountCls**

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

**get_set**() → int

```
# SCPI: DIAGnostic:INFO:POCount:SET
value: int = driver.diagnostic.info.poCount.get_set()
```

No command help available

**return**
set_py: No help available

**get_value**() → int

```
# SCPI: DIAGnostic:INFO:POCount
value: int = driver.diagnostic.info.poCount.get_value()
```

Queris how often the instrument has been turned on so far.

**return**
power_on_count: integer Range: 0 to INT_MAX

**set_set**(*set_py: int*) → None

```
# SCPI: DIAGnostic:INFO:POCount:SET
driver.diagnostic.info.poCount.set_set(set_py = 1)
```

No command help available

**param set_py**
No help available

### 6.5.5 Measure

**class MeasureCls**

Measure 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.diagnostic.measure.clone()
```

#### Subgroups

##### 6.5.5.1 Point

#### SCPI Command :

```
DIAGnostic<HW>:[MEASure]:POINt
```

**class PointCls**

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

**get**(*name: str*) → str

```
# SCPI: DIAGnostic<HW>:[MEASure]:POINt
value: str = driver.diagnostic.measure.point.get(name = 'abc')
```

Triggers the voltage measurement at the specified test point and returns the measured voltage. For more information, see R&S SMA100B Service Manual.

**param name**
test point identifier Test point name, as queried with the command method RsSmab.Diagnostic.Point.catalog

**return**
value: valueunit

### 6.5.6 Point

#### SCPI Command :

```
DIAGnostic<HW>:POINt:CATalog
```

**class PointCls**

Point commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_catalog**() → List[str]

```
# SCPI: DIAGnostic<HW>:POINt:CATalog
value: List[str] = driver.diagnostic.point.get_catalog()
```

Queries the test points available in the instrument. For more information, see R&S SMA100B Service Manual.

**return**
catalog: string List of comma-separated values, each representing a test point

### Cloning the Group

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

### Subgroups

#### 6.5.6.1 Configuration

### SCPI Command :

```
DIAGnostic<HW>:POINt:CONFiguration
```

**class ConfigurationCls**

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

**class GetStruct**

Response structure. Fields:

- Dev_Board: str: No parameter help available
- Point: str: No parameter help available

**get**() → GetStruct

```
# SCPI: DIAGnostic<HW>:POINt:CONFiguration
value: GetStruct = driver.diagnostic.point.configuration.get()
```

No command help available

**return**
structure: for return value, see the help for GetStruct structure arguments.

**set**(*dev_board: str*, *point: str*, *data: str*) → None

```
# SCPI: DIAGnostic<HW>:POINt:CONFiguration
driver.diagnostic.point.configuration.set(dev_board = 'abc', point = 'abc',␣
↪data = 'abc')
```

No command help available

**param dev_board**
No help available

**param point**
No help available

**param data**
No help available

### 6.5.7 Service

#### SCPI Commands :

```
DIAGnostic<HW>:SERVice:SFUNction
DIAGnostic:SERVice
```

**class ServiceCls**

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

**get_sfunction**() → str

```
# SCPI: DIAGnostic<HW>:SERVice:SFUNction
value: str = driver.diagnostic.service.get_sfunction()
```

No command help available

**return**
direct_string: No help available

**get_value**() → bool

```
# SCPI: DIAGnostic:SERVice
value: bool = driver.diagnostic.service.get_value()
```

No command help available

**return**
service: No help available

**set_sfunction**(*direct_string: str*) → None

```
# SCPI: DIAGnostic<HW>:SERVice:SFUNction
driver.diagnostic.service.set_sfunction(direct_string = 'abc')
```

No command help available

**param direct_string**
No help available

**set_value**(*service: bool*) → None

```
# SCPI: DIAGnostic:SERVice
driver.diagnostic.service.set_value(service = False)
```

No command help available

**param service**
No help available

## 6.6 Display

### SCPI Commands :

```
DISPlay:BRIGhtness
DISPlay:FOCusobject
DISPlay:MESSage
```

**class DisplayCls**

Display commands group definition. 21 total commands, 8 Subgroups, 3 group commands

**get_brightness**() → float

```
# SCPI: DISPlay:BRIGhtness
value: float = driver.display.get_brightness()
```

Sets the brightness of the dispaly.

**return**
brightness: float Range: 1.0 to 20.0

**set_brightness**(*brightness: float*) → None

```
# SCPI: DISPlay:BRIGhtness
driver.display.set_brightness(brightness = 1.0)
```

Sets the brightness of the dispaly.

**param brightness**
float Range: 1.0 to 20.0

**set_focus_object**(*obj_name: str*) → None

```
# SCPI: DISPlay:FOCusobject
driver.display.set_focus_object(obj_name = 'abc')
```

No command help available

**param obj_name**
No help available

**set_message**(*message: str*) → None

```
# SCPI: DISPlay:MESSage
driver.display.set_message(message = 'abc')
```

No command help available

**param message**
No help available

#### Cloning the Group

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

#### Subgroups

### 6.6.1 Annotation

#### SCPI Command :

```
DISPlay:ANNotation:[ALL]
```

**class AnnotationCls**

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

**get_all**() → bool

```
# SCPI: DISPlay:ANNotation:[ALL]
value: bool = driver.display.annotation.get_all()
```

Displays asterisks instead of the level and frequency values in the status bar of the instrument. We recommend that you use this mode if you operate the instrument in remote control.

**return**
state: 1| ON| 0| OFF

**set_all**(*state: bool*) → None

```
# SCPI: DISPlay:ANNotation:[ALL]
driver.display.annotation.set_all(state = False)
```

Displays asterisks instead of the level and frequency values in the status bar of the instrument. We recommend that you use this mode if you operate the instrument in remote control.

**param state**
1| ON| 0| OFF

#### Cloning the Group

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

### Subgroups

#### 6.6.1.1 Amplitude

### SCPI Command :

```
DISPlay:ANNotation:AMPLitude
```

**class AmplitudeCls**

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

**class AmplitudeStruct**

Response structure. Fields:

- Sec_Pass_Word: str: No parameter help available
- State: bool: 1| ON| 0| OFF

**get**() → AmplitudeStruct

```
# SCPI: DISPlay:ANNotation:AMPLitude
value: AmplitudeStruct = driver.display.annotation.amplitude.get()
```

Indicates asterisks instead of the level values in the status bar.

**return**
structure: for return value, see the help for AmplitudeStruct structure arguments.

**set**(*sec_pass_word: str*, *state: bool*) → None

```
# SCPI: DISPlay:ANNotation:AMPLitude
driver.display.annotation.amplitude.set(sec_pass_word = 'abc', state = False)
```

Indicates asterisks instead of the level values in the status bar.

**param sec_pass_word**
No help available

**param state**
1| ON| 0| OFF

#### 6.6.1.2 Frequency

### SCPI Command :

```
DISPlay:ANNotation:FREQuency
```

**class FrequencyCls**

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

**class FrequencyStruct**

Response structure. Fields:

- Sec_Pass_Word: str: No parameter help available
- State: bool: 1| ON| 0| OFF

**get**() → FrequencyStruct

```
# SCPI: DISPlay:ANNotation:FREQuency
value: FrequencyStruct = driver.display.annotation.frequency.get()
```

Indicates asterisks instead of the frequency values in the status bar.

**return**
structure: for return value, see the help for FrequencyStruct structure arguments.

**set**(*sec_pass_word: str*, *state: bool*) → None

```
# SCPI: DISPlay:ANNotation:FREQuency
driver.display.annotation.frequency.set(sec_pass_word = 'abc', state = False)
```

Indicates asterisks instead of the frequency values in the status bar.

**param sec_pass_word**
No help available

**param state**
1| ON| 0| OFF

### 6.6.2 Button

#### SCPI Command :

```
DISPlay:BUTTon:BRIGhtness
```

**class ButtonCls**

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

**get_brightness**() → int

```
# SCPI: DISPlay:BUTTon:BRIGhtness
value: int = driver.display.button.get_brightness()
```

Sets the brightness of the [RF on/off] key.

**return**
button_brightnes: integer Range: 1 to 20

**set_brightness**(*button_brightnes: int*) → None

```
# SCPI: DISPlay:BUTTon:BRIGhtness
driver.display.button.set_brightness(button_brightnes = 1)
```

Sets the brightness of the [RF on/off] key.

**param button_brightnes**
integer Range: 1 to 20

### 6.6.3 Dialog

#### SCPI Commands :

```
DISPlay:DIALog:CLOSe
DISPlay:DIALog:CLOSe:ALL
DISPlay:DIALog:ID
DISPlay:DIALog:OPEN
```

**class DialogCls**

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

**close**(*dialog_id: str*) → None

```
# SCPI: DISPlay:DIALog:CLOSe
driver.display.dialog.close(dialog_id = 'abc')
```

Closes the specified dialog.

**param dialog_id**
string To find out the dialog identifier, use the query method RsSmab.Display.Dialog.id. The DialogName part of the query result is sufficient.

**close_all**() → None

```
# SCPI: DISPlay:DIALog:CLOSe:ALL
driver.display.dialog.close_all()
```

Closes all open dialogs.

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

```
# SCPI: DISPlay:DIALog:CLOSe:ALL
driver.display.dialog.close_all_with_opc()
```

Closes all open dialogs.

Same as close_all, but waits for the operation to complete before continuing further. Use the RsSmab.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_id**() → str

```
# SCPI: DISPlay:DIALog:ID
value: str = driver.display.dialog.get_id()
```

Returns the dialog identifiers of the open dialogs in a string separated by blanks.

**return**
dialog_id_list: DialogID#1 DialogID#2 ... DialogID#n Dialog identifiers are string without blanks. Blanks are represented as $$. Dialog identifiers DialogID are composed of two main parts: DialogName[OptionalParts] DialogName Meaningful information, mandatory input parameter for the commands: method RsSmab.Display.Dialog.open method RsSmab.Display.Dialog.close Optional parts String of $X values, where X is a character, interpreted as follows: $qDialogQualifier:

optional dialog qualifier, usually the letter A or B, as displayed in the dialog title. $iInstances: comma-separated list of instance indexes, given in the order h,c,s,d,g,u,0. Default is zero; the terminating ',0' can be omitted. $tTabIds: comma-separated indexes or tab names; required, if a dialog is composed of several tabs. $xLeft$yTop$hLeft$wTop: position and size; superfluous information.

**open**(*dialog_id: str*) → None

```
# SCPI: DISPlay:DIALog:OPEN
driver.display.dialog.open(dialog_id = 'abc')
```

Opens the specified dialog.

**param dialog_id**
string To find out the dialog identifier, use the query method RsSmab.Display.Dialog.id. The DialogName part of the query result is mandatory.

### 6.6.4 Psave

#### SCPI Commands :

```
DISPlay:PSAVe:HOLDoff
DISPlay:PSAVe:[STATe]
```

**class PsaveCls**

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

**get_holdoff**() → int

```
# SCPI: DISPlay:PSAVe:HOLDoff
value: int = driver.display.psave.get_holdoff()
```

Sets the wait time for the screen saver mode of the display.

**return**
holdoff_time_min: integer Range: 1 to 60, Unit: minute

**get_state**() → bool

```
# SCPI: DISPlay:PSAVe:[STATe]
value: bool = driver.display.psave.get_state()
```

Activates the screen saver mode of the display. We recommend that you use this mode to protect the display, if you operate the instrument in remote control. To define the wait time, use the command method RsSmab.Display.Psave.holdoff.

**return**
state: 1| ON| 0| OFF

**set_holdoff**(*holdoff_time_min: int*) → None

```
# SCPI: DISPlay:PSAVe:HOLDoff
driver.display.psave.set_holdoff(holdoff_time_min = 1)
```

Sets the wait time for the screen saver mode of the display.

**param holdoff_time_min**
integer Range: 1 to 60, Unit: minute

**set_state**(*state: bool*) → None

```
# SCPI: DISPlay:PSAVe:[STATe]
driver.display.psave.set_state(state = False)
```

Activates the screen saver mode of the display. We recommend that you use this mode to protect the display, if you operate the instrument in remote control. To define the wait time, use the command method RsSmab.Display.Psave.holdoff.

**param state**
1| ON| 0| OFF

### 6.6.5 Touch

**class TouchCls**

Touch 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.display.touch.clone()
```

#### Subgroups

##### 6.6.5.1 Time

#### SCPI Command :

```
DISPlay:TOUCh:TIME:CHARge
```

**class TimeCls**

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

**set_charge**(*charge_time: int*) → None

```
# SCPI: DISPlay:TOUCh:TIME:CHARge
driver.display.touch.time.set_charge(charge_time = 1)
```

No command help available

**param charge_time**
No help available

### 6.6.6 Ukey

#### SCPI Commands :

```
DISPlay:UKEY:NAME
DISPlay:UKEY:SCPI
```

**class UkeyCls**

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

**set_name**(*name: str*) → None

```
# SCPI: DISPlay:UKEY:NAME
driver.display.ukey.set_name(name = 'abc')
```

No command help available

**param name**
No help available

**set_scpi**(*scpi: str*) → None

```
# SCPI: DISPlay:UKEY:SCPI
driver.display.ukey.set_scpi(scpi = 'abc')
```

No command help available

**param scpi**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.6.6.1 Add

#### SCPI Command :

```
DISPlay:UKEY:ADD
```

**class AddCls**

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

**set**() → None

```
# SCPI: DISPlay:UKEY:ADD
driver.display.ukey.add.set()
```

No command help available

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

```
# SCPI: DISPlay:UKEY:ADD
driver.display.ukey.add.set_with_opc()
```

No command help available

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

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

### 6.6.7 Update

#### SCPI Commands :

```
DISPlay:UPDate:HOLD
DISPlay:UPDate:[STATe]
```

**class UpdateCls**

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

**get_hold**() → bool

```
# SCPI: DISPlay:UPDate:HOLD
value: bool = driver.display.update.get_hold()
```

No command help available

**return**
hold: No help available

**get_state**() → bool

```
# SCPI: DISPlay:UPDate:[STATe]
value: bool = driver.display.update.get_state()
```

Activates the refresh mode of the display.

**return**
update: 1| ON| 0| OFF

**set_hold**(*hold: bool*) → None

```
# SCPI: DISPlay:UPDate:HOLD
driver.display.update.set_hold(hold = False)
```

No command help available

**param hold**
No help available

**set_state**(*update: bool*) → None

```
# SCPI: DISPlay:UPDate:[STATe]
driver.display.update.set_state(update = False)
```

Activates the refresh mode of the display.

**param update**
1| ON| 0| OFF

### 6.6.8 Window

**class WindowCls**

Window 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.display.window.clone()
```

#### Subgroups

##### 6.6.8.1 Power

**class PowerCls**

Power 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.display.window.power.clone()
```

#### Subgroups

##### 6.6.8.1.1 Sweep

**class SweepCls**

Sweep 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.display.window.power.sweep.clone()
```

#### Subgroups

##### 6.6.8.1.1.1 Background

#### SCPI Command :

```
DISPlay:[WINDow]:[POWer]:SWEep:BACKground:COLor
```

**class BackgroundCls**

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

**get_color**() → DiagBgColor

```
# SCPI: DISPlay:[WINDow]:[POWer]:SWEep:BACKground:COLor
value: enums.DiagBgColor = driver.display.window.power.sweep.background.get_
↪color()
```

Defines the background color of the measurement diagram. The selected color applies also to the hardcopy of the diagram.

**return**
color: BLACk| WHITe

**set_color**(*color: DiagBgColor*) → None

```
# SCPI: DISPlay:[WINDow]:[POWer]:SWEep:BACKground:COLor
driver.display.window.power.sweep.background.set_color(color = enums.
↪DiagBgColor.BLACk)
```

Defines the background color of the measurement diagram. The selected color applies also to the hardcopy of the diagram.

**param color**
BLACk| WHITe

##### 6.6.8.1.2 Grid

#### SCPI Command :

```
DISPlay:[WINDow]:[POWer]:SWEep:GRID:STATe
```

**class GridCls**

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

**get_state**() → bool

```
# SCPI: DISPlay:[WINDow]:[POWer]:SWEep:GRID:STATe
value: bool = driver.display.window.power.sweep.grid.get_state()
```

Indicates a grid in the diagram.

**return**
state: 0| 1| OFF| ON

**set_state**(*state: bool*) → None

```
# SCPI: DISPlay:[WINDow]:[POWer]:SWEep:GRID:STATe
driver.display.window.power.sweep.grid.set_state(state = False)
```

Indicates a grid in the diagram.

**param state**
0| 1| OFF| ON

## 6.7 FormatPy

### SCPI Commands :

```
FORMat:BORDer
FORMat:SREGister
FORMat:[DATA]
```

**class FormatPyCls**

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

**get_border**() → ByteOrder

```
# SCPI: FORMat:BORDer
value: enums.ByteOrder = driver.formatPy.get_border()
```

Determines the sequence of bytes within a binary block. This only affects blocks which use the IEEE754 format internally.

**return**
border: NORMal| SWAPped NORMal Expects/sends the least significant byte of each IEEE754 floating-point number first and the most significant byte last. SWAPped Expects/sends the most significant byte of each IEEE754 floating-point number first and the least significant byte last.

**get_data**() → FormData

```
# SCPI: FORMat:[DATA]
value: enums.FormData = driver.formatPy.get_data()
```

Determines the data format the instrument uses to return data via the IEC/IEEE bus. The instrument automatically detects the data format used by the controller, and assigns it accordingly. Data format determined by this SCPI command is in this case irrelevant.

**return**
data: ASCii| PACKed ASCii Transfers numerical data as plain text separated by commas. PACKed Transfers numerical data as binary block data. The format within the binary data depends on the command. The various binary data formats are explained in the description of the parameter types.

**get_sregister**() → FormStatReg

```
# SCPI: FORMat:SREGister
value: enums.FormStatReg = driver.formatPy.get_sregister()
```

Determines the numeric format for responses of the status register.

**return**
format_py: ASCii| BINary| HEXadecimal| OCTal ASCii Returns the register content as a decimal number. BINary|HEXadecimal|OCTal Returns the register content either as a binary, hexadecimal or octal number. According to the selected format, the number starts with #B (binary) , #H (hexadecimal) or #O (octal) .

**set_border**(*border: ByteOrder*) → None

```
# SCPI: FORMat:BORDer
driver.formatPy.set_border(border = enums.ByteOrder.NORMal)
```

Determines the sequence of bytes within a binary block. This only affects blocks which use the IEEE754 format internally.

**param border**
NORMal| SWAPped NORMal Expects/sends the least significant byte of each IEEE754 floating-point number first and the most significant byte last. SWAPped Expects/sends the most significant byte of each IEEE754 floating-point number first and the least significant byte last.

**set_data**(*data: FormData*) → None

```
# SCPI: FORMat:[DATA]
driver.formatPy.set_data(data = enums.FormData.ASCii)
```

Determines the data format the instrument uses to return data via the IEC/IEEE bus. The instrument automatically detects the data format used by the controller, and assigns it accordingly. Data format determined by this SCPI command is in this case irrelevant.

**param data**
ASCii| PACKed ASCii Transfers numerical data as plain text separated by commas. PACKed Transfers numerical data as binary block data. The format within the binary data depends on the command. The various binary data formats are explained in the description of the parameter types.

**set_sregister**(*format_py: FormStatReg*) → None

```
# SCPI: FORMat:SREGister
driver.formatPy.set_sregister(format_py = enums.FormStatReg.ASCii)
```

Determines the numeric format for responses of the status register.

**param format_py**
ASCii| BINary| HEXadecimal| OCTal ASCii Returns the register content as a decimal number. BINary|HEXadecimal|OCTal Returns the register content either as a binary, hexadecimal or octal number. According to the selected format, the number starts with #B (binary) , #H (hexadecimal) or #O (octal) .

## 6.8 Fpanel

**class FpanelCls**

Fpanel 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.fpanel.clone()
```

### Subgroups

### 6.8.1 Keyboard

#### SCPI Command :

```
FPANel:KEYBoard:LAYout
```

**class KeyboardCls**

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

**get_layout**() → FrontPanelLayout

```
# SCPI: FPANel:KEYBoard:LAYout
value: enums.FrontPanelLayout = driver.fpanel.keyboard.get_layout()
```

No command help available

**return**
layout: No help available

**set_layout**(*layout: FrontPanelLayout*) → None

```
# SCPI: FPANel:KEYBoard:LAYout
driver.fpanel.keyboard.set_layout(layout = enums.FrontPanelLayout.DIGits)
```

No command help available

**param layout**
No help available

## 6.9 HardCopy

#### SCPI Commands :

```
HCOPy:DATA
HCOPy:REGion
```

**class HardCopyCls**

HardCopy commands group definition. 17 total commands, 4 Subgroups, 2 group commands

**get_data**() → bytes

```
# SCPI: HCOPy:DATA
value: bytes = driver.hardCopy.get_data()
```

Transfers the hard copy data directly as a NByte stream to the remote client.

**return**
data: block data

**get_region**() → HardCopyRegion

```
# SCPI: HCOPy:REGion
value: enums.HardCopyRegion = driver.hardCopy.get_region()
```

Selects the area to be copied. You can create a snapshot of the screen or an active dialog.

**return**
region: ALL| DIALog

**set_region**(*region: HardCopyRegion*) → None

```
# SCPI: HCOPy:REGion
driver.hardCopy.set_region(region = enums.HardCopyRegion.ALL)
```

Selects the area to be copied. You can create a snapshot of the screen or an active dialog.

**param region**
ALL| DIALog

#### Cloning the Group

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

#### Subgroups

### 6.9.1 Device

#### SCPI Command :

```
HCOPy:DEVice:LANGuage
```

**class DeviceCls**

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

**get_language**() → HardCopyImageFormat

```
# SCPI: HCOPy:DEVice:LANGuage
value: enums.HardCopyImageFormat = driver.hardCopy.device.get_language()
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**return**
language: BMP| JPG| XPM| PNG

**set_language**(*language: HardCopyImageFormat*) → None

```
# SCPI: HCOPy:DEVice:LANGuage
driver.hardCopy.device.set_language(language = enums.HardCopyImageFormat.BMP)
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**param language**
BMP| JPG| XPM| PNG

### 6.9.2 Execute

#### SCPI Command :

```
HCOPy:[EXECute]
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: HCOPy:[EXECute]
driver.hardCopy.execute.set()
```

Generates a hard copy of the current display. The output destination is a file.

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

```
# SCPI: HCOPy:[EXECute]
driver.hardCopy.execute.set_with_opc()
```

Generates a hard copy of the current display. The output destination is a file.

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

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

### 6.9.3 File

**class FileCls**

File commands group definition. 12 total commands, 1 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.9.3.1 Name

### SCPI Command :

```
HCOPy:FILE:[NAME]
```

**class NameCls**

Name commands group definition. 12 total commands, 1 Subgroups, 1 group commands

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]
value: str = driver.hardCopy.file.name.get_value()
```

Determines the file name and path to save the hard copy, provided automatic naming is disabled. Note: If you have enabled automatic naming, the instrument automatically generates the file name and directory, see 'Automatic naming'.

**return**
name: string

**set_value**(*name: str*) → None

```
# SCPI: HCOPy:FILE:[NAME]
driver.hardCopy.file.name.set_value(name = 'abc')
```

Determines the file name and path to save the hard copy, provided automatic naming is disabled. Note: If you have enabled automatic naming, the instrument automatically generates the file name and directory, see 'Automatic naming'.

**param name**
string

### Cloning the Group

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

### Subgroups

#### 6.9.3.1.1 Auto

### SCPI Commands :

```
HCOPy:FILE:[NAME]:AUTO:STATe
HCOPy:FILE:[NAME]:AUTO
```

**class AutoCls**

Auto commands group definition. 11 total commands, 2 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:STATe
value: bool = driver.hardCopy.file.name.auto.get_state()
```

Activates automatic naming of the hard copy files.

**return**
state: 1| ON| 0| OFF

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]:AUTO
value: str = driver.hardCopy.file.name.auto.get_value()
```

Queries path and file name of the hardcopy file, if you have enabled Automatic Naming.

**return**
auto: string

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:STATe
driver.hardCopy.file.name.auto.set_state(state = False)
```

Activates automatic naming of the hard copy files.

**param state**
1| ON| 0| OFF

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.hardCopy.file.name.auto.clone()
```

### Subgroups

#### 6.9.3.1.1.1 Directory

### SCPI Commands :

```
HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
HCOPy:FILE:[NAME]:AUTO:DIRectory
```

**class DirectoryCls**

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

**clear**() → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
driver.hardCopy.file.name.auto.directory.clear()
```

Deletes all files with extensions *.bmp, *.jpg, *.png and *.xpm in the directory set for automatic naming.

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

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
driver.hardCopy.file.name.auto.directory.clear_with_opc()
```

Deletes all files with extensions *.bmp, *.jpg, *.png and *.xpm in the directory set for automatic naming.

Same as clear, but waits for the operation to complete before continuing further. Use the RsSmab.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: HCOPy:FILE:[NAME]:AUTO:DIRectory
value: str = driver.hardCopy.file.name.auto.directory.get_value()
```

Determines the path to save the hard copy, if you have enabled Automatic Naming. If the directory does not yet exist, the instrument automatically creates a new directory, using the instrument name and /var/user/ by default.

**return**
directory: string

**set_value**(*directory: str*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:DIRectory
driver.hardCopy.file.name.auto.directory.set_value(directory = 'abc')
```

Determines the path to save the hard copy, if you have enabled Automatic Naming. If the directory does not yet exist, the instrument automatically creates a new directory, using the instrument name and /var/user/ by default.

**param directory**
string

#### 6.9.3.1.1.2 File

##### SCPI Commands :

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:NUMBer
HCOPy:FILE:[NAME]:AUTO:FILE
```

**class FileCls**

File commands group definition. 7 total commands, 4 Subgroups, 2 group commands

**get_number**() → int

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:NUMBer
value: int = driver.hardCopy.file.name.auto.file.get_number()
```

Queries the number that is used as part of the file name for the next hard copy in automatic mode. At the beginning, the count starts at 0. The R&S SMA100B searches the specified output directory for the highest number in the stored files. It increases this number by one to achieve a unique name for the new file. The resulting auto number is appended to the resulting file name with at least three digits.

**return**
number: integer Range: 0 to 999999

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:FILE
value: str = driver.hardCopy.file.name.auto.file.get_value()
```

Queries the name of the automatically named hard copy file. An automatically generated file name consists of: <Prefix><YYYY><MM><DD><Number>.<Format>. You can activate each component separately, to individually design the file name.

**return**
file: string

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.hardCopy.file.name.auto.file.clone()
```

##### Subgroups

#### 6.9.3.1.1.3 Day

##### SCPI Command :

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
```

**class DayCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
value: bool = driver.hardCopy.file.name.auto.file.day.get_state()
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
driver.hardCopy.file.name.auto.file.day.set_state(state = False)
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**param state**
1| ON| 0| OFF

#### 6.9.3.1.1.4 Month

**SCPI Command :**

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
```

**class MonthCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
value: bool = driver.hardCopy.file.name.auto.file.month.get_state()
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
driver.hardCopy.file.name.auto.file.month.set_state(state = False)
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**param state**
1| ON| 0| OFF

#### 6.9.3.1.1.5 Prefix

**SCPI Commands :**

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
```

**class PrefixCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
value: bool = driver.hardCopy.file.name.auto.file.prefix.get_state()
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**return**
state: 1| ON| 0| OFF

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
value: str = driver.hardCopy.file.name.auto.file.prefix.get_value()
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**return**
prefix: No help available

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
driver.hardCopy.file.name.auto.file.prefix.set_state(state = False)
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**param state**
1| ON| 0| OFF

**set_value**(*prefix: str*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
driver.hardCopy.file.name.auto.file.prefix.set_value(prefix = 'abc')
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**param prefix**
1| ON| 0| OFF

#### 6.9.3.1.1.6 Year

**SCPI Command :**

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
```

**class YearCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
value: bool = driver.hardCopy.file.name.auto.file.year.get_state()
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
driver.hardCopy.file.name.auto.file.year.set_state(state = False)
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**param state**
1| ON| 0| OFF

### 6.9.4 Image

**SCPI Command :**

```
HCOPy:IMAGe:FORMat
```

**class ImageCls**

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

**get_format_py**() → HardCopyImageFormat

```
# SCPI: HCOPy:IMAGe:FORMat
value: enums.HardCopyImageFormat = driver.hardCopy.image.get_format_py()
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**return**
format_py: No help available

**set_format_py**(*format_py: HardCopyImageFormat*) → None

```
# SCPI: HCOPy:IMAGe:FORMat
driver.hardCopy.image.set_format_py(format_py = enums.HardCopyImageFormat.BMP)
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**param format_py**
BMP| JPG| XPM| PNG

## 6.10 Initiate<Channel>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.initiate.repcap_channel_get()
driver.initiate.repcap_channel_set(repcap.Channel.Nr1)
```

**class InitiateCls**

Initiate commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

### Cloning the Group

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

### Subgroups

### 6.10.1 FreqSweep

**class FreqSweepCls**

FreqSweep 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.initiate.freqSweep.clone()
```

### Subgroups

#### 6.10.1.1 Continuous

### SCPI Command :

```
INITiate<HW>:FSWeep:CONTinuous
```

**class ContinuousCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: INITiate<HW>:FSWeep:CONTinuous
value: bool = driver.initiate.freqSweep.continuous.get(channel = repcap.Channel.
↪Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

**return**
freq_sweep_state: No help available

**set**(*freq_sweep_state: bool*, *channel=Channel.Default*) → None

```
# SCPI: INITiate<HW>:FSWeep:CONTinuous
driver.initiate.freqSweep.continuous.set(freq_sweep_state = False, channel =␣
↪repcap.Channel.Default)
```

No command help available

**param freq_sweep_state**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

### 6.10.2 LffSweep

**class LffSweepCls**

LffSweep 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.initiate.lffSweep.clone()
```

#### Subgroups

##### 6.10.2.1 Continuous

#### SCPI Command :

```
INITiate<HW>:LFFSweep:CONTinuous
```

**class ContinuousCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: INITiate<HW>:LFFSweep:CONTinuous
value: bool = driver.initiate.lffSweep.continuous.get(channel = repcap.Channel.
↪Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

**return**
sw_lf_init_state: No help available

**set**(*sw_lf_init_state: bool*, *channel=Channel.Default*) → None

```
# SCPI: INITiate<HW>:LFFSweep:CONTinuous
driver.initiate.lffSweep.continuous.set(sw_lf_init_state = False, channel =␣
↪repcap.Channel.Default)
```

No command help available

**param sw_lf_init_state**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

### 6.10.3 ListPy

**class ListPyCls**

ListPy 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.initiate.listPy.clone()
```

#### Subgroups

##### 6.10.3.1 Mode

#### SCPI Command :

```
INITiate:LIST:MODE:CONTinuous
```

**class ModeCls**

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

**get_continuous**() → bool

```
# SCPI: INITiate:LIST:MODE:CONTinuous
value: bool = driver.initiate.listPy.mode.get_continuous()
```

No command help available

**return**
list_mode_adv_stat: No help available

**set_continuous**(*list_mode_adv_stat: bool*) → None

```
# SCPI: INITiate:LIST:MODE:CONTinuous
driver.initiate.listPy.mode.set_continuous(list_mode_adv_stat = False)
```

No command help available

**param list_mode_adv_stat**
No help available

### 6.10.4 Power

**class PowerCls**

Power 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.initiate.power.clone()
```

#### Subgroups

##### 6.10.4.1 Continuous

#### SCPI Command :

```
INITiate<HW>:[POWer]:CONTinuous
```

**class ContinuousCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: INITiate<HW>:[POWer]:CONTinuous
value: bool = driver.initiate.power.continuous.get(channel = repcap.Channel.
↪Default)
```

Switches the local state of the continuous power measurement by R&S NRP power sensors on and off. Switching off local state enhances the measurement performance during remote control. The remote measurement is triggered with method RsSmab. **Read.Power.get_**) . This command also returns the measurement results. The local state is not affected, measurement results can be retrieved with local state on or off.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

**return**
continuous: 1| ON| 0| OFF

**set**(*continuous: bool*, *channel=Channel.Default*) → None

```
# SCPI: INITiate<HW>:[POWer]:CONTinuous
driver.initiate.power.continuous.set(continuous = False, channel = repcap.
↪Channel.Default)
```

Switches the local state of the continuous power measurement by R&S NRP power sensors on and off. Switching off local state enhances the measurement performance during remote control. The remote measurement is triggered with method RsSmab. **Read.Power.get_**) . This command also returns the measurement results. The local state is not affected, measurement results can be retrieved with local state on or off.

**param continuous**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

## 6.10.5 Psweep

**class PsweepCls**

Psweep 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.initiate.psweep.clone()
```

### Subgroups

#### 6.10.5.1 Continuous

### SCPI Command :

```
INITiate<HW>:PSWeep:CONTinuous
```

**class ContinuousCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: INITiate<HW>:PSWeep:CONTinuous
value: bool = driver.initiate.psweep.continuous.get(channel = repcap.Channel.
↪Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

**return**
sw_pow_init_state: No help available

**set**(*sw_pow_init_state: bool*, *channel=Channel.Default*) → None

```
# SCPI: INITiate<HW>:PSWeep:CONTinuous
driver.initiate.psweep.continuous.set(sw_pow_init_state = False, channel =␣
↪repcap.Channel.Default)
```

No command help available

**param sw_pow_init_state**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

## 6.11 Kboard

### SCPI Command :

```
KBOard:LAYout
```

**class KboardCls**

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

**get_layout**() → KbLayout

```
# SCPI: KBOard:LAYout
value: enums.KbLayout = driver.kboard.get_layout()
```

Selects the language for an external keyboard and assigns the keys acccordingly.

**return**
layout: CHINese| DANish| DUTCh| DUTBe| ENGLish| ENGUK| FINNish| FRENch| FREBe| FRECa| GERMan| ITALian| JAPanese| KORean| NORWegian| PORTuguese| RUSSian| SPANish| SWEDish| ENGUS

**set_layout**(*layout: KbLayout*) → None

```
# SCPI: KBOard:LAYout
driver.kboard.set_layout(layout = enums.KbLayout.CHINese)
```

Selects the language for an external keyboard and assigns the keys acccordingly.

**param layout**
CHINese| DANish| DUTCh| DUTBe| ENGLish| ENGUK| FINNish| FRENch| FREBe| FRECa| GERMan| ITALian| JAPanese| KORean| NORWegian| PORTuguese| RUSSian| SPANish| SWEDish| ENGUS

## 6.12 MassMemory

### SCPI Commands :

```
MMEMory:CDIRectory
MMEMory:COPY
MMEMory:DELete
MMEMory:DRIVes
MMEMory:MDIRectory
MMEMory:MOVE
MMEMory:MSIS
MMEMory:RDIRectory
MMEMory:RDIRectory:RECursive
```

**class MassMemoryCls**

MassMemory commands group definition. 15 total commands, 4 Subgroups, 9 group commands

**copy**(*source_file: str*, *destination_file: str*) → None

```
# SCPI: MMEMory:COPY
driver.massMemory.copy(source_file = 'abc', destination_file = 'abc')
```

Copies an existing file to a new file. Instead of just a file, this command can also be used to copy a complete directory together with all its files.

**param source_file**
string String containing the path and file name of the source file

**param destination_file**
string String containing the path and name of the target file. The path can be relative or absolute. If DestinationFile is not specified, the SourceFile is copied to the current directory, queried with the method RsSmab.MassMemory.currentDirectory command. Note: Existing files with the same name in the destination directory are overwritten without an error message.

**delete**(*filename: str*) → None

```
# SCPI: MMEMory:DELete
driver.massMemory.delete(filename = 'abc')
```

Removes a file from the specified directory.

**param filename**
string String parameter to specify the name and directory of the file to be removed.

**delete_directory**(*directory: str*) → None

```
# SCPI: MMEMory:RDIRectory
driver.massMemory.delete_directory(directory = 'abc')
```

Removes an existing directory from the mass memory storage system. If no directory is specified, the subdirectory with the specified name is deleted in the default directory.

**param directory**
string String parameter to specify the directory to be deleted.

**delete_directory_recursive**(*directory: str*) → None

```
# SCPI: MMEMory:RDIRectory:RECursive
driver.massMemory.delete_directory_recursive(directory = 'abc')
```

No command help available

**param directory**
No help available

**get_current_directory**() → str

```
# SCPI: MMEMory:CDIRectory
value: str = driver.massMemory.get_current_directory()
```

Changes the default directory for mass memory storage. The directory is used for all subsequent MMEM commands if no path is specified with them.

**return**
directory: directory_name String containing the path to another directory. The path can be relative or absolute. To change to a higher directory, use two dots '..' .

**get_drives**() → str

```
# SCPI: MMEMory:DRIVes
value: str = driver.massMemory.get_drives()
```

No command help available

**return**
drive_list: No help available

**get_msis**() → str

```
# SCPI: MMEMory:MSIS
value: str = driver.massMemory.get_msis()
```

Defines the drive or network resource (in the case of networks) for instruments with windows operating system, using msis (MSIS = Mass Storage Identification String) . Note: Instruments with Linux operating system ignore this command, since Linux does not use drive letter assignment.

**return**
path: No help available

**make_directory**(*directory: str*) → None

```
# SCPI: MMEMory:MDIRectory
driver.massMemory.make_directory(directory = 'abc')
```

Creates a subdirectory for mass memory storage in the specified directory. If no directory is specified, a subdirectory is created in the default directory. This command can also be used to create a directory tree.

**param directory**
string String parameter to specify the new directory.

**move**(*source_file: str*, *destination_file: str*) → None

```
# SCPI: MMEMory:MOVE
driver.massMemory.move(source_file = 'abc', destination_file = 'abc')
```

Moves an existing file to a new location or, if no path is specified, renames an existing file.

**param source_file**
string String parameter to specify the name of the file to be moved.

**param destination_file**
string String parameters to specify the name of the new file.

**set_current_directory**(*directory: str*) → None

```
# SCPI: MMEMory:CDIRectory
driver.massMemory.set_current_directory(directory = 'abc')
```

Changes the default directory for mass memory storage. The directory is used for all subsequent MMEM commands if no path is specified with them.

**param directory**
directory_name String containing the path to another directory. The path can be relative or absolute. To change to a higher directory, use two dots '..' .

**set_msis**(*path: str*) → None

```
# SCPI: MMEMory:MSIS
driver.massMemory.set_msis(path = 'abc')
```

Defines the drive or network resource (in the case of networks) for instruments with windows operating system, using msis (MSIS = Mass Storage Identification String) . Note: Instruments with Linux operating system ignore this command, since Linux does not use drive letter assignment.

**param path**
No help available

#### Cloning the Group

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

#### Subgroups

### 6.12.1 Catalog

#### SCPI Command :

```
MMEMory:CATalog
```

**class CatalogCls**

Catalog commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_value**() → str

```
# SCPI: MMEMory:CATalog
value: str = driver.massMemory.catalog.get_value()
```

Returns the content of a particular directory.

**return**
catalog: No help available

#### Cloning the Group

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

#### Subgroups

##### 6.12.1.1 Length

#### SCPI Command :

```
MMEMory:CATalog:LENGth
```

**class LengthCls**

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

**get**(*path: str = None*) → int

```
# SCPI: MMEMory:CATalog:LENGth
value: int = driver.massMemory.catalog.length.get(path = 'abc')
```

Returns the number of files in the current or in the specified directory.

**param path**
string String parameter to specify the directory. If the directory is omitted, the command queries the content of the current directory, queried with method RsSmab.MassMemory.currentDirectory command.

**return**
file_count: integer Number of files.

### 6.12.2 Dcatalog

#### SCPI Command :

```
MMEMory:DCATalog
```

**class DcatalogCls**

Dcatalog commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_value**() → str

```
# SCPI: MMEMory:DCATalog
value: str = driver.massMemory.dcatalog.get_value()
```

Returns the subdirectories of a particular directory.

**return**
dcatalog: No help available

#### Cloning the Group

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

#### Subgroups

##### 6.12.2.1 Length

#### SCPI Command :

```
MMEMory:DCATalog:LENGth
```

**class LengthCls**

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

**get**(*path: str = None*) → int

```
# SCPI: MMEMory:DCATalog:LENGth
value: int = driver.massMemory.dcatalog.length.get(path = 'abc')
```

Returns the number of subdirectories in the current or specified directory.

**param path**
String parameter to specify the directory. If the directory is omitted, the command queries the contents of the current directory, to be queried with method RsSmab.MassMemory.currentDirectory command.

**return**
directory_count: integer Number of parent and subdirectories.

### 6.12.3 Load

**class LoadCls**

Load 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.massMemory.load.clone()
```

#### Subgroups

##### 6.12.3.1 State

#### SCPI Command :

```
MMEMory:LOAD:STATe
```

**class StateCls**

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

**set**(*data_set: int*, *source_file: str*) → None

```
# SCPI: MMEMory:LOAD:STATe
driver.massMemory.load.state.set(data_set = 1, source_file = 'abc')
```

Loads the specified file stored under the specified name in an internal memory. After the file has been loaded, the instrument setting must be activated using an *RCL command.

**param data_set**
No help available

**param source_file**
No help available

### 6.12.4 Store

**class StoreCls**

Store 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.massMemory.store.clone()
```

#### Subgroups

##### 6.12.4.1 State

#### SCPI Command :

```
MMEMory:STORe:STATe
```

**class StateCls**

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

**set**(*data_set: int*, *destination_file: str*) → None

```
# SCPI: MMEMory:STORe:STATe
driver.massMemory.store.state.set(data_set = 1, destination_file = 'abc')
```

Stores the current instrument setting in the specified file. The instrument setting must first be stored in an internal memory with the same number using the common command *SAV.

**param data_set**
No help available

**param destination_file**
No help available

## 6.13 Memory

### SCPI Command :

```
MEMory:HFRee
```

**class MemoryCls**

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

**class HfreeStruct**

Structure for reading output parameters. Fields:

- Total_Phys_Mem_Kb: List[int]: integer Total physical memory.
- Applic_Mem_Kb: int: integer Application memory.
- Heap_Used_Kb: int: integer Used heap memory.
- Heap_Available_Kb: int: integer Available heap memory.

**get_hfree()** → HfreeStruct

```
# SCPI: MEMory:HFRee
value: HfreeStruct = driver.memory.get_hfree()
```

Returns the used and available memory in Kb.

**return**
structure: for return value, see the help for HfreeStruct structure arguments.

## 6.14 Output

### SCPI Commands :

```
OUTPut<HW>:AMODe
OUTPut<HW>:IMPedance
```

**class OutputCls**

Output commands group definition. 12 total commands, 7 Subgroups, 2 group commands

**get_amode()** → PowAttModeOut

```
# SCPI: OUTPut<HW>:AMODe
value: enums.PowAttModeOut = driver.output.get_amode()
```

Sets the step attenuator mode at the RF output.

**return**
amode: AUTO| FIXed AUTO The step attenuator adjusts the level settings automatically, within the full variation range. FIXed The step attenuator and amplifier stages are fixed at the current position, providing level settings with constant output VSWR. The resulting variation range is calculated according to the position.

**get_impedance**() → InpImpRf

```
# SCPI: OUTPut<HW>:IMPedance
value: enums.InpImpRf = driver.output.get_impedance()
```

Queries the impedance of the RF outputs.

**return**
impedance: G1K| G50| G10K

**set_amode**(*amode: PowAttModeOut*) → None

```
# SCPI: OUTPut<HW>:AMODe
driver.output.set_amode(amode = enums.PowAttModeOut.AUTO)
```

Sets the step attenuator mode at the RF output.

**param amode**
AUTO| FIXed AUTO The step attenuator adjusts the level settings automatically, within the full variation range. FIXed The step attenuator and amplifier stages are fixed at the current position, providing level settings with constant output VSWR. The resulting variation range is calculated according to the position.

#### Cloning the Group

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

#### Subgroups

### 6.14.1 Afixed

**class AfixedCls**

Afixed 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.output.afixed.clone()
```

#### Subgroups

##### 6.14.1.1 Range

#### SCPI Commands :

```
OUTPut<HW>:AFIXed:RANGe:LOWer
OUTPut<HW>:AFIXed:RANGe:UPPer
```

**class RangeCls**

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

**get_lower**() → float

```
# SCPI: OUTPut<HW>:AFIXed:RANGe:LOWer
value: float = driver.output.afixed.range.get_lower()
```

Queries the settable minimum/maximum value in mode OUTPut:AMODe FIXed, i.e. when the attenuator is not being adjusted. See method RsSmab.Output.amode

**return**
lower: float Unit: dBm

**get_upper**() → float

```
# SCPI: OUTPut<HW>:AFIXed:RANGe:UPPer
value: float = driver.output.afixed.range.get_upper()
```

Queries the settable minimum/maximum value in mode OUTPut:AMODe FIXed, i.e. when the attenuator is not being adjusted. See method RsSmab.Output.amode

**return**
upper: float Unit: dBm

### 6.14.2 All

#### SCPI Command :

```
OUTPut:ALL:[STATe]
```

**class AllCls**

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

**get_state**() → bool

```
# SCPI: OUTPut:ALL:[STATe]
value: bool = driver.output.all.get_state()
```

Activates the RF output signal of the instrument.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: OUTPut:ALL:[STATe]
driver.output.all.set_state(state = False)
```

Activates the RF output signal of the instrument.

**param state**
1| ON| 0| OFF

### 6.14.3 FilterPy

#### SCPI Command :

```
OUTPut<HW>:FILTer:MODE
```

**class FilterPyCls**

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

**get_mode**() → PowHarmMode

```
# SCPI: OUTPut<HW>:FILTer:MODE
value: enums.PowHarmMode = driver.output.filterPy.get_mode()
```

Activates low harmonic filter or enables its automatic switching.

**return**
mode: ON| AUTO| 1 ON|1 Ensures best low harmonics performance but decreases the level range AUTO Applies an automatically selected harmonic filter that fits to the current level setting.

**set_mode**(*mode: PowHarmMode*) → None

```
# SCPI: OUTPut<HW>:FILTer:MODE
driver.output.filterPy.set_mode(mode = enums.PowHarmMode._1)
```

Activates low harmonic filter or enables its automatic switching.

**param mode**
ON| AUTO| 1 ON|1 Ensures best low harmonics performance but decreases the level range AUTO Applies an automatically selected harmonic filter that fits to the current level setting.

### 6.14.4 Fproportional

#### SCPI Command :

```
OUTPut:FPRoportional:SCALe
```

**class FproportionalCls**

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

**get_scale**() → SelOutpVxAxis

```
# SCPI: OUTPut:FPRoportional:SCALe
value: enums.SelOutpVxAxis = driver.output.fproportional.get_scale()
```

Selects the mode the voltage is supplied depending on the frequency. The R&S SMA100B supplies the signal at the V/GHz X-Axis connector.

**return**
outp_sel_scale: S0V25| S0V5| S1V0| XAXis S0V25|S0V5|S1V0 Supplies the voltage proportional to the set frequency, derived from the selected setting. XAXis Supplies a voltage range from 0 V to 10 V proportional to the frequency sweep range, set with[:SOURcehw]:FREQuency:STARt and [:SOURcehw]:FREQuency:STOP.

**set_scale**(*outp_sel_scale: SelOutpVxAxis*) → None

```
# SCPI: OUTPut:FPRoportional:SCALe
driver.output.fproportional.set_scale(outp_sel_scale = enums.SelOutpVxAxis.
↪S0V25)
```

Selects the mode the voltage is supplied depending on the frequency. The R&S SMA100B supplies the signal at the V/GHz X-Axis connector.

**param outp_sel_scale**
S0V25| S0V5| S1V0| XAXis S0V25|S0V5|S1V0 Supplies the voltage proportional to the set frequency, derived from the selected setting. XAXis Supplies a voltage range from 0 V to 10 V proportional to the frequency sweep range, set with[:SOURcehw]:FREQuency:STARt and [:SOURcehw]:FREQuency:STOP.

### 6.14.5 Protection

#### SCPI Commands :

```
OUTPut<HW>:PROTection:CLEar
OUTPut<HW>:PROTection:TRIPped
```

**class ProtectionCls**

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

**clear**() → None

```
# SCPI: OUTPut<HW>:PROTection:CLEar
driver.output.protection.clear()
```

Resets the protective circuit after it has been tripped. To define the output state, use the command method RsSmab.Output. State.value.

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

```
# SCPI: OUTPut<HW>:PROTection:CLEar
driver.output.protection.clear_with_opc()
```

Resets the protective circuit after it has been tripped. To define the output state, use the command method RsSmab.Output. State.value.

Same as clear, but waits for the operation to complete before continuing further. Use the RsSmab.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_tripped**() → bool

```
# SCPI: OUTPut<HW>:PROTection:TRIPped
value: bool = driver.output.protection.get_tripped()
```

Queries the state of the protective circuit.

**return**
tripped: 1| ON| 0| OFF

### 6.14.6 State

#### SCPI Commands :

```
OUTPut<HW>:[STATe]:PON
OUTPut<HW>:[STATe]
```

**class StateCls**

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

**get_pon**() → UnchOff

```
# SCPI: OUTPut<HW>:[STATe]:PON
value: enums.UnchOff = driver.output.state.get_pon()
```

Defines the state of the RF output signal when the instrument is switched on.

**return**
pon: OFF| UNCHanged

**get_value**() → bool

```
# SCPI: OUTPut<HW>:[STATe]
value: bool = driver.output.state.get_value()
```

Activates the RF output signal.

**return**
state: 1| ON| 0| OFF

**set_pon**(*pon: UnchOff*) → None

```
# SCPI: OUTPut<HW>:[STATe]:PON
driver.output.state.set_pon(pon = enums.UnchOff.OFF)
```

Defines the state of the RF output signal when the instrument is switched on.

**param pon**
OFF| UNCHanged

**set_value**(*state: bool*) → None

```
# SCPI: OUTPut<HW>:[STATe]
driver.output.state.set_value(state = False)
```

Activates the RF output signal.

**param state**
1| ON| 0| OFF

### 6.14.7 User

#### SCPI Command :

```
OUTPut:USER:MARKer
```

**class UserCls**

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

**get_marker**() → SelOutpMarkUser

```
# SCPI: OUTPut:USER:MARKer
value: enums.SelOutpMarkUser = driver.output.user.get_marker()
```

Selects the signal for output at the Marker User1 connector.

**return**
sel_user_marker: MARK| USER MARK Assigns a marker signal to the output. USER Intended for future use.

**set_marker**(*sel_user_marker: SelOutpMarkUser*) → None

```
# SCPI: OUTPut:USER:MARKer
driver.output.user.set_marker(sel_user_marker = enums.SelOutpMarkUser.MARK)
```

Selects the signal for output at the Marker User1 connector.

**param sel_user_marker**
MARK| USER MARK Assigns a marker signal to the output. USER Intended for future use.

## 6.15 Read<Channel>

#### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.read.repcap_channel_get()
driver.read.repcap_channel_set(repcap.Channel.Nr1)
```

**class ReadCls**

Read commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

#### Cloning the Group

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

#### Subgroups

### 6.15.1 Power

#### SCPI Command :

```
READ<CH>:[POWer]
```

**class PowerCls**

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

**get**(*channel=Channel.Default*) → List[float]

```
# SCPI: READ<CH>:[POWer]
value: List[float] = driver.read.power.get(channel = repcap.Channel.Default)
```

Triggers power measurement and displays the results. Note: This command does not affect the local state, i.e. you can get results with local state on or off. For long measurement times, we recommend that you use an SRQ for command synchronization (MAV bit) .

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Read')

**return**
power: float or float,float The sensor returns the result in the unit set with command method RsSmab.Sense.Unit.Power.set Certain power sensors, such as the R&S NRP-Z81, return two values, first the value of the average level and - separated by a comma - the peak value.

## 6.16 Sense<Channel>

#### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.sense.repcap_channel_get()
driver.sense.repcap_channel_set(repcap.Channel.Nr1)
```

**class SenseCls**

Sense commands group definition. 120 total commands, 2 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

#### Cloning the Group

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

#### Subgroups

### 6.16.1 Power

**class PowerCls**

Power commands group definition. 119 total commands, 15 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.16.1.1 Aperture

**class ApertureCls**

Aperture 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.sense.power.aperture.clone()
```

#### Subgroups

###### 6.16.1.1.1 Default

**class DefaultCls**

Default 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.sense.power.aperture.default.clone()
```

#### Subgroups

##### 6.16.1.1.1.1 State

#### SCPI Command :

```
SENSe<CH>:[POWer]:APERture:DEFault:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:APERture:DEFault:STATe
value: bool = driver.sense.power.aperture.default.state.get(channel = repcap.
↪Channel.Default)
```

Deactivates the default aperture time of the respective sensor. To specify a user-defined value, use the command method RsSmab.Sense.Power.Aperture.Time.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
use_def_ap: 1| ON| 0| OFF

**set**(*use_def_ap: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:APERture:DEFault:STATe
driver.sense.power.aperture.default.state.set(use_def_ap = False, channel =␣
↪repcap.Channel.Default)
```

Deactivates the default aperture time of the respective sensor. To specify a user-defined value, use the command method RsSmab.Sense.Power.Aperture.Time.set.

**param use_def_ap**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

##### 6.16.1.1.2 Time

#### SCPI Command :

```
SENSe<CH>:[POWer]:APERture:TIMe
```

**class TimeCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:APERture:TIMe
value: float = driver.sense.power.aperture.time.get(channel = repcap.Channel.
↪Default)
```

Defines the aperture time (size of the acquisition interval) for the corresponding sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
ap_time: float Range: depends on connected power sensor

**set**(*ap_time: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:APERture:TIMe
driver.sense.power.aperture.time.set(ap_time = 1.0, channel = repcap.Channel.
↪Default)
```

Defines the aperture time (size of the acquisition interval) for the corresponding sensor.

**param ap_time**
float Range: depends on connected power sensor

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.2 Correction

**class CorrectionCls**

Correction 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.sense.power.correction.clone()
```

#### Subgroups

##### 6.16.1.2.1 SpDevice

**class SpDeviceCls**

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

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.correction.spDevice.clone()
```

#### Subgroups

##### 6.16.1.2.1.1 ListPy

#### SCPI Command :

```
SENSe<CH>:[POWer]:CORRection:SPDevice:LIST
```

**class ListPyCls**

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

**get**(*channel=Channel.Default*) → List[str]

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:LIST
value: List[str] = driver.sense.power.correction.spDevice.listPy.get(channel =␣
↪repcap.Channel.Default)
```

Queries the list of the S-parameter data sets that have been loaded to the power sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
list_py: string list

##### 6.16.1.2.1.2 Select

#### SCPI Command :

```
SENSe<CH>:[POWer]:CORRection:SPDevice:SELect
```

**class SelectCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:SELect
value: float = driver.sense.power.correction.spDevice.select.get(channel =␣
↪repcap.Channel.Default)
```

Several S-parameter tables can be stored in a sensor. The command selects a loaded data set for S-parameter correction for the corresponding sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
select: float

**set**(*select: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:SELect
driver.sense.power.correction.spDevice.select.set(select = 1.0, channel =␣
↪repcap.Channel.Default)
```

Several S-parameter tables can be stored in a sensor. The command selects a loaded data set for S-parameter correction for the corresponding sensor.

**param select**
float

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.2.1.3 State

#### SCPI Command :

```
SENSe<CH>:[POWer]:CORRection:SPDevice:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:STATe
value: bool = driver.sense.power.correction.spDevice.state.get(channel = repcap.
↪Channel.Default)
```

Activates the use of the S-parameter correction data. Note: If you use power sensors with attenuator, the instrument automatically activates the use of S-parameter data.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:STATe
driver.sense.power.correction.spDevice.state.set(state = False, channel =␣
↪repcap.Channel.Default)
```

Activates the use of the S-parameter correction data. Note: If you use power sensors with attenuator, the instrument automatically activates the use of S-parameter data.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.3 Direct

#### SCPI Command :

```
SENSe<CH>:[POWer]:DIRect
```

**class DirectCls**

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

**set**(*command: str*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:DIRect
driver.sense.power.direct.set(command = 'abc', channel = repcap.Channel.Default)
```

No command help available

**param command**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.4 Display

**class DisplayCls**

Display 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.sense.power.display.clone()
```

#### Subgroups

##### 6.16.1.4.1 Permanent

**class PermanentCls**

Permanent 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.sense.power.display.permanent.clone()
```

### Subgroups

#### 6.16.1.4.1.1 Priority

### SCPI Command :

```
SENSe<CH>:[POWer]:DISPlay:PERManent:PRIority
```

**class PriorityCls**

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

**get**(*channel=Channel.Default*) → PowSensDisplayPriority

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:PRIority
value: enums.PowSensDisplayPriority = driver.sense.power.display.permanent.
↪priority.get(channel = repcap.Channel.Default)
```

Selects average or peak power for permanent display.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
priority: AVERage| PEAK

**set**(*priority: PowSensDisplayPriority*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:PRIority
driver.sense.power.display.permanent.priority.set(priority = enums.
↪PowSensDisplayPriority.AVERage, channel = repcap.Channel.Default)
```

Selects average or peak power for permanent display.

**param priority**
AVERage| PEAK

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.4.1.2 State

**SCPI Command :**

```
SENSe<CH>:[POWer]:DISPlay:PERManent:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:STATe
value: bool = driver.sense.power.display.permanent.state.get(channel = repcap.
↪Channel.Default)
```

Activates the permanent display of the measured power level results. The instrument also indicates the sensor type, the connection, the measurement source and the offset if set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:STATe
driver.sense.power.display.permanent.state.set(state = False, channel = repcap.
↪Channel.Default)
```

Activates the permanent display of the measured power level results. The instrument also indicates the sensor type, the connection, the measurement source and the offset if set.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### 6.16.1.5 FilterPy

**class FilterPyCls**

FilterPy commands group definition. 6 total commands, 4 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.16.1.5.1 Length

**class LengthCls**

Length 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.sense.power.filterPy.length.clone()
```

#### Subgroups

##### 6.16.1.5.1.1 Auto

#### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:LENGth:AUTO
```

**class AutoCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:LENGth:AUTO
value: float = driver.sense.power.filterPy.length.auto.get(channel = repcap.
↪Channel.Default)
```

Queries the current filter length in filter mode AUTO (method RsSmab.Sense.Power.FilterPy.TypePy.set)

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
auto: float Range: 1 to 65536

#### 6.16.1.5.1.2 User

**SCPI Command :**

```
SENSe<CH>:[POWer]:FILTer:LENGth:[USER]
```

**class UserCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:LENGth:[USER]
value: float = driver.sense.power.filterPy.length.user.get(channel = repcap.
↪Channel.Default)
```

Selects the filter length for SENS:POW:FILT:'TYPE USER. As the filter length works as a multiplier for the time window, a constant filter length results in a constant measurement time (see also 'About the measuring principle, averaging filter, filter length, and achieving stable results') .

INTRO_CMD_HELP: The R&S NRP power sensors provide different resolutions for setting the filter length, depending on the used sensor type:

- Resolution = 1 for R&S NRPxx power sensors
- Resolution = 2n for sensors of the R&S NRP-Zxx family, with n = 1 to 16

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
user: float Range: 1 to 65536

**set**(*user: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:LENGth:[USER]
driver.sense.power.filterPy.length.user.set(user = 1.0, channel = repcap.
↪Channel.Default)
```

Selects the filter length for SENS:POW:FILT:'TYPE USER. As the filter length works as a multiplier for the time window, a constant filter length results in a constant measurement time (see also 'About the measuring principle, averaging filter, filter length, and achieving stable results') .

INTRO_CMD_HELP: The R&S NRP power sensors provide different resolutions for setting the filter length, depending on the used sensor type:

- Resolution = 1 for R&S NRPxx power sensors
- Resolution = 2n for sensors of the R&S NRP-Zxx family, with n = 1 to 16

**param user**
float Range: 1 to 65536

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.5.2 NsRatio

### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:NSRatio
```

**class NsRatioCls**

NsRatio commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio
value: float = driver.sense.power.filterPy.nsRatio.get(channel = repcap.Channel.
↪Default)
```

Sets an upper limit for the relative noise content in fixed noise filter mode (method RsSmab.Sense.Power.FilterPy.TypePy. set) . This value determines the proportion of intrinsic noise in the measurement results.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
ns_ratio: float Range: 0.001 to 1

**set**(*ns_ratio: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio
driver.sense.power.filterPy.nsRatio.set(ns_ratio = 1.0, channel = repcap.
↪Channel.Default)
```

Sets an upper limit for the relative noise content in fixed noise filter mode (method RsSmab.Sense.Power.FilterPy.TypePy. set) . This value determines the proportion of intrinsic noise in the measurement results.

**param ns_ratio**
float Range: 0.001 to 1

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.filterPy.nsRatio.clone()
```

#### Subgroups

##### 6.16.1.5.2.1 Mtime

#### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:NSRatio:MTIMe
```

**class MtimeCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio:MTIMe
value: float = driver.sense.power.filterPy.nsRatio.mtime.get(channel = repcap.
↪Channel.Default)
```

Sets an upper limit for the settling time of the auto-averaging filter in the NSRatio mode and thus limits the length of the filter. The filter type is set with command method RsSmab.Sense.Power.FilterPy.TypePy.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
mtime: float Range: 1 to 999.99

**set**(*mtime: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio:MTIMe
driver.sense.power.filterPy.nsRatio.mtime.set(mtime = 1.0, channel = repcap.
↪Channel.Default)
```

Sets an upper limit for the settling time of the auto-averaging filter in the NSRatio mode and thus limits the length of the filter. The filter type is set with command method RsSmab.Sense.Power.FilterPy.TypePy.set.

**param mtime**
float Range: 1 to 999.99

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

##### 6.16.1.5.3 Sonce

#### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:SONCe
```

**class SonceCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:SONCe
driver.sense.power.filterPy.sonce.set(channel = repcap.Channel.Default)
```

Starts searching the optimum filter length for the current measurement conditions. You can check the result with command SENS1:POW:FILT:LENG:USER? in filter mode USER (method RsSmab.Sense.Power.FilterPy.TypePy.set) .

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.5.4 TypePy

#### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:TYPE
```

**class TypePyCls**

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

**get**(*channel=Channel.Default*) → PowSensFiltType

```
# SCPI: SENSe<CH>:[POWer]:FILTer:TYPE
value: enums.PowSensFiltType = driver.sense.power.filterPy.typePy.get(channel =␣
↪repcap.Channel.Default)
```

Selects the filter mode. The filter length is the multiplier for the time window and thus directly affects the measurement time.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
type_py: AUTO| USER| NSRatio AUTO Automatically selects the filter length, depending on the measured value. The higher the power, the shorter the filter length, and vice versa. USER Allows you to set the filter length manually. As the filter-length takes effect as a multiplier of the measurement time, you can achieve constant measurement times. NSRatio Selects the filter length (averaging factor) according to the criterion that the intrinsic noise of the sensor (2 standard deviations) does not exceed the specified noise content. You can define the noise content with command method RsSmab.Sense.Power.FilterPy.NsRatio.set. Note: To avoid long settling times when the power is low, you can limit the averaging factor limited with the 'timeout' parameter (method RsSmab.Sense.Power.FilterPy.NsRatio.Mtime.set) .

**set**(*type_py: PowSensFiltType*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:TYPE
driver.sense.power.filterPy.typePy.set(type_py = enums.PowSensFiltType.AUTO,␣
↪channel = repcap.Channel.Default)
```

Selects the filter mode. The filter length is the multiplier for the time window and thus directly affects the measurement time.

**param type_py**
AUTO| USER| NSRatio AUTO Automatically selects the filter length, depending on the measured value. The higher the power, the shorter the filter length, and vice versa. USER Allows you to set the filter length manually. As the filter-length takes effect as a multiplier of the measurement time, you can achieve constant measurement times. NSRatio Selects the filter length (averaging factor) according to the criterion that the intrinsic noise of the sensor (2 standard deviations) does not exceed the specified noise content. You can define the noise content with command method RsSmab.Sense.Power.FilterPy.NsRatio.set. Note: To avoid long settling times when the power is low, you can limit the averaging factor limited with the 'timeout' parameter (method RsSmab.Sense.Power.FilterPy.NsRatio.Mtime.set) .

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.6 Frequency

#### SCPI Command :

```
SENSe<CH>:[POWer]:FREQuency
```

**class FrequencyCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FREQuency
value: float = driver.sense.power.frequency.get(channel = repcap.Channel.
↪Default)
```

Sets the RF frequency of the signal, if signal source SENSe<ch>[:POWer]:SOURce USER is selected.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
frequency: float

**set**(*frequency: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FREQuency
driver.sense.power.frequency.set(frequency = 1.0, channel = repcap.Channel.
↪Default)
```

Sets the RF frequency of the signal, if signal source SENSe<ch>[:POWer]:SOURce USER is selected.

**param frequency**
float

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.7 Logging

**class LoggingCls**

Logging 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.sense.power.logging.clone()
```

### Subgroups

#### 6.16.1.7.1 State

### SCPI Command :

```
SENSe<CH>:[POWer]:LOGGing:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:LOGGing:STATe
value: bool = driver.sense.power.logging.state.get(channel = repcap.Channel.
↪Default)
```

Activates the recording of the power values, measured by a connected R&S NRP power sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:LOGGing:STATe
driver.sense.power.logging.state.set(state = False, channel = repcap.Channel.
↪Default)
```

Activates the recording of the power values, measured by a connected R&S NRP power sensor.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.8 Offset

**SCPI Command :**

```
SENSe<CH>:[POWer]:OFFSet
```

**class OffsetCls**

Offset commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:OFFSet
value: float = driver.sense.power.offset.get(channel = repcap.Channel.Default)
```

Sets a level offset which is added to the measured level value after activation with command method RsSmab.Sense.Power. Offset.State.set. The level offset allows, e.g. to consider an attenuator in the signal path.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
offset: float Range: -100.0 to 100.0, Unit: dB

**set**(*offset: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:OFFSet
driver.sense.power.offset.set(offset = 1.0, channel = repcap.Channel.Default)
```

Sets a level offset which is added to the measured level value after activation with command method RsSmab.Sense.Power. Offset.State.set. The level offset allows, e.g. to consider an attenuator in the signal path.

**param offset**
float Range: -100.0 to 100.0, Unit: dB

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**Cloning the Group**

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

#### Subgroups

##### 6.16.1.8.1 State

#### SCPI Command :

```
SENSe<CH>:[POWer]:OFFSet:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:OFFSet:STATe
value: bool = driver.sense.power.offset.state.get(channel = repcap.Channel.
↪Default)
```

Activates the addition of the level offset to the measured value. The level offset value is set with command method RsSmab.Sense.Power.Offset.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:OFFSet:STATe
driver.sense.power.offset.state.set(state = False, channel = repcap.Channel.
↪Default)
```

Activates the addition of the level offset to the measured value. The level offset value is set with command method RsSmab.Sense.Power.Offset.set.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

##### 6.16.1.9 Snumber

#### SCPI Command :

```
SENSe<CH>:[POWer]:SNUMber
```

**class SnumberCls**

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

**get**(*channel=Channel.Default*) → str

```
# SCPI: SENSe<CH>:[POWer]:SNUMber
value: str = driver.sense.power.snumber.get(channel = repcap.Channel.Default)
```

Queries the serial number of the sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
snumber: string

#### 6.16.1.10 Source

#### SCPI Command :

```
SENSe<CH>:[POWer]:SOURce
```

**class SourceCls**

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

**get**(*channel=Channel.Default*) → PowSensSource

```
# SCPI: SENSe<CH>:[POWer]:SOURce
value: enums.PowSensSource = driver.sense.power.source.get(channel = repcap.
↪Channel.Default)
```

Determines the signal to be measured. Note: When measuring the RF signal, the sensor considers the corresponding correction factor at that frequency, and uses the level setting of the instrument as reference level.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
source: A| USER| RF

**set**(*source: PowSensSource*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SOURce
driver.sense.power.source.set(source = enums.PowSensSource.A, channel = repcap.
↪Channel.Default)
```

Determines the signal to be measured. Note: When measuring the RF signal, the sensor considers the corresponding correction factor at that frequency, and uses the level setting of the instrument as reference level.

**param source**
A| USER| RF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.11 Status

**class StatusCls**

Status 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.sense.power.status.clone()
```

### Subgroups

#### 6.16.1.11.1 Device

### SCPI Command :

```
SENSe<CH>:[POWer]:STATus:[DEVice]
```

**class DeviceCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:STATus:[DEVice]
value: bool = driver.sense.power.status.device.get(channel = repcap.Channel.
↪Default)
```

Queries if a sensor is connected to the instrument.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
status: 1| ON| 0| OFF

#### 6.16.1.12 Sversion

### SCPI Command :

```
SENSe<CH>:[POWer]:SVERsion
```

**class SversionCls**

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

**get**(*channel=Channel.Default*) → str

```
# SCPI: SENSe<CH>:[POWer]:SVERsion
value: str = driver.sense.power.sversion.get(channel = repcap.Channel.Default)
```

Queries the software version of the connected R&S NRP power sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
sversion: string

#### 6.16.1.13 Sweep

**SCPI Commands :**

```
SENSe:[POWer]:SWEep:ABORt
SENSe:[POWer]:SWEep:INITiate
SENSe:[POWer]:SWEep:MODE
SENSe:[POWer]:SWEep:RMODe
```

**class SweepCls**

Sweep commands group definition. 95 total commands, 4 Subgroups, 4 group commands

**abort**(*rf_pow_sens_meas_resp_meas_event: bool*) → None

```
# SCPI: SENSe:[POWer]:SWEep:ABORt
driver.sense.power.sweep.abort(rf_pow_sens_meas_resp_meas_event = False)
```

Aborts the power analysis with NRP power sensors.

**param rf_pow_sens_meas_resp_meas_event**
No help available

**get_mode**() → MeasRespMode

```
# SCPI: SENSe:[POWer]:SWEep:MODE
value: enums.MeasRespMode = driver.sense.power.sweep.get_mode()
```

Selects power versus frequency measurement (frequency response) , power vs power measurement (power sweep, AM/AM) or power vs. time measurement.

**return**
mode: FREQuency| POWer| TIME

**get_rmode**() → RepeatMode

```
# SCPI: SENSe:[POWer]:SWEep:RMODe
value: enums.RepeatMode = driver.sense.power.sweep.get_rmode()
```

Selects single or continuous mode for power analysis (all measurement modes) .

**return**
rmode: SINGle| CONTinuous

**initiate**(*rf_pow_sens_meas_resp_meas_event: bool*) → None

```
# SCPI: SENSe:[POWer]:SWEep:INITiate
driver.sense.power.sweep.initiate(rf_pow_sens_meas_resp_meas_event = False)
```

Starts the power analysis with NRP power sensor.

**param rf_pow_sens_meas_resp_meas_event**
No help available

**set_mode**(*mode: MeasRespMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:MODE
driver.sense.power.sweep.set_mode(mode = enums.MeasRespMode.FREQuency)
```

Selects power versus frequency measurement (frequency response) , power vs power measurement (power sweep, AM/AM) or power vs. time measurement.

**param mode**
FREQuency| POWer| TIME

**set_rmode**(*rmode: RepeatMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:RMODe
driver.sense.power.sweep.set_rmode(rmode = enums.RepeatMode.CONTinuous)
```

Selects single or continuous mode for power analysis (all measurement modes) .

**param rmode**
SINGle| CONTinuous

### Cloning the Group

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

### Subgroups

#### 6.16.1.13.1 Frequency

### SCPI Commands :

```
SENSe:[POWer]:SWEep:FREQuency:RMODe
SENSe:[POWer]:SWEep:FREQuency:STARt
SENSe:[POWer]:SWEep:FREQuency:STEPs
SENSe:[POWer]:SWEep:FREQuency:STOP
```

**class FrequencyCls**

Frequency commands group definition. 19 total commands, 5 Subgroups, 4 group commands

**get_rmode**() → RepeatMode

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:RMODe
value: enums.RepeatMode = driver.sense.power.sweep.frequency.get_rmode()
```

Selects single or continuous mode for measurement mode frequency in power analysis.

**return**
rmode: SINGle| CONTinuous

**get_start**() → float

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:STARt
value: float = driver.sense.power.sweep.frequency.get_start()
```

Sets the start frequency for the frequency mode.

**return**
start: float Range: 0 to 1E12

**get_steps**() → int

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:STEPs
value: int = driver.sense.power.sweep.frequency.get_steps()
```

Sets the number of measurement steps for the frequency mode.

**return**
steps: integer Range: 10 to 1000

**get_stop**() → float

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:STOP
value: float = driver.sense.power.sweep.frequency.get_stop()
```

Sets the stop frequency for the frequency mode.

**return**
stop: float Range: 0 to 1E12

**set_rmode**(*rmode: RepeatMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:RMODe
driver.sense.power.sweep.frequency.set_rmode(rmode = enums.RepeatMode.
↪CONTinuous)
```

Selects single or continuous mode for measurement mode frequency in power analysis.

**param rmode**
SINGle| CONTinuous

**set_start**(*start: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:STARt
driver.sense.power.sweep.frequency.set_start(start = 1.0)
```

Sets the start frequency for the frequency mode.

**param start**
float Range: 0 to 1E12

**set_steps**(*steps: int*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:STEPs
driver.sense.power.sweep.frequency.set_steps(steps = 1)
```

Sets the number of measurement steps for the frequency mode.

**param steps**
integer Range: 10 to 1000

**set_stop**(*stop: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:STOP
driver.sense.power.sweep.frequency.set_stop(stop = 1.0)
```

Sets the stop frequency for the frequency mode.

**param stop**
float Range: 0 to 1E12

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.frequency.clone()
```

### Subgroups

#### 6.16.1.13.1.1 Reference

**class ReferenceCls**

Reference 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.sense.power.sweep.frequency.reference.clone()
```

### Subgroups

#### 6.16.1.13.1.2 Data

### SCPI Commands :

```
SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:COPY
SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:POINts
SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:XVALues
SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:YVALues
```

**class DataCls**

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

**copy**() → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:COPY
driver.sense.power.sweep.frequency.reference.data.copy()
```

Generates a reference curve for 'Frequency' measurement.

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

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:COPY
driver.sense.power.sweep.frequency.reference.data.copy_with_opc()
```

Generates a reference curve for 'Frequency' measurement.

Same as copy, but waits for the operation to complete before continuing further. Use the RsSmab.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_points**() → int

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:POINts
value: int = driver.sense.power.sweep.frequency.reference.data.get_points()
```

Queries the number of points from the reference curve in 'Frequency' measurement.

**return**
points: integer Range: 10 to 1000

**get_xvalues**() → List[float]

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:XVALues
value: List[float] = driver.sense.power.sweep.frequency.reference.data.get_
↪xvalues()
```

Sets the x values of the two reference points, i.e. 'Frequency X (Point A) ' and 'Frequency X (Point B) ' in 'Frequency' measurement.

**return**
xvalues: string

**get_yvalues**() → List[float]

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:YVALues
value: List[float] = driver.sense.power.sweep.frequency.reference.data.get_
↪yvalues()
```

Sets or queries the y values of the two reference points, i.e.'Pow Y (Point A) ' and 'Power Y (Point B) ' in 'Frequency' measurement.

**return**
yvalues: string

**set_xvalues**(*xvalues: List[float]*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:XVALues
driver.sense.power.sweep.frequency.reference.data.set_xvalues(xvalues = [1.1, 2.
↪2, 3.3])
```

Sets the x values of the two reference points, i.e. 'Frequency X (Point A) ' and 'Frequency X (Point B) ' in 'Frequency' measurement.

**param xvalues**
string

**set_yvalues**(*yvalues: List[float]*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:REFerence:DATA:YVALues
driver.sense.power.sweep.frequency.reference.data.set_yvalues(yvalues = [1.1, 2.
↪2, 3.3])
```

Sets or queries the y values of the two reference points, i.e.'Pow Y (Point A) ' and 'Power Y (Point B) ' in 'Frequency' measurement.

**param yvalues**
string

#### 6.16.1.13.1.3 Sensor

**class SensorCls**

Sensor commands group definition. 5 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.frequency.sensor.clone()
```

### Subgroups

#### 6.16.1.13.1.4 Offset

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:OFFSet
```

**class OffsetCls**

Offset commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:OFFSet
value: float = driver.sense.power.sweep.frequency.sensor.offset.get(channel =␣
↪repcap.Channel.Default)
```

Defines the level offset at the sensor input in dB. Activate the offset with the command method RsSmab.Sense.Power.Sweep. Frequency.Sensor.Offset.State.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
offset: float Range: -100 to 100

**set**(*offset: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:OFFSet
driver.sense.power.sweep.frequency.sensor.offset.set(offset = 1.0, channel =␣
↪repcap.Channel.Default)
```

Defines the level offset at the sensor input in dB. Activate the offset with the command method RsSmab.Sense.Power.Sweep. Frequency.Sensor.Offset.State.set.

**param offset**
float Range: -100 to 100

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.frequency.sensor.offset.clone()
```

### Subgroups

#### 6.16.1.13.1.5 State

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:OFFSet:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:OFFSet:STATe
value: bool = driver.sense.power.sweep.frequency.sensor.offset.state.
↪get(channel = repcap.Channel.Default)
```

Activates a level offset at the sensor input. Define the appropriate value with the command method RsSmab.Sense.Power. Sweep.Frequency.Sensor.Offset.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:OFFSet:STATe
driver.sense.power.sweep.frequency.sensor.offset.state.set(state = False,␣
↪channel = repcap.Channel.Default)
```

Activates a level offset at the sensor input. Define the appropriate value with the command method RsSmab.Sense.Power. Sweep.Frequency.Sensor.Offset.set.

**param state**
0| 1| OFF| ON

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.1.6 Srange

**class SrangeCls**

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

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.frequency.sensor.srange.clone()
```

### Subgroups

#### 6.16.1.13.1.7 Start

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:STARt
```

**class StartCls**

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

**get**(*channel=Channel.Default*) → int

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:STARt
value: int = driver.sense.power.sweep.frequency.sensor.srange.start.get(channel
↪= repcap.Channel.Default)
```

Sets the start frequency for the frequency power analysis with separate frequencies.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
start: integer Range: 0 to 1E12

**set**(*start: int*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:STARt
driver.sense.power.sweep.frequency.sensor.srange.start.set(start = 1, channel =
↪repcap.Channel.Default)
```

Sets the start frequency for the frequency power analysis with separate frequencies.

**param start**
integer Range: 0 to 1E12

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.1.8 State

#### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:[STATe]
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:[STATe]
value: bool = driver.sense.power.sweep.frequency.sensor.srange.state.
↪get(channel = repcap.Channel.Default)
```

Activates the use of a frequency range for the power measurement that is different to the set signal generator frequency range. The separate frequency range is entered with commands method RsSmab.Sense.Power.Sweep.Frequency.Sensor.Srange. Start.set and method RsSmab.Sense.Power.Sweep.Frequency.Sensor.Srange.Stop.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:[STATe]
driver.sense.power.sweep.frequency.sensor.srange.state.set(state = False,␣
↪channel = repcap.Channel.Default)
```

Activates the use of a frequency range for the power measurement that is different to the set signal generator frequency range. The separate frequency range is entered with commands method RsSmab.Sense.Power.Sweep.Frequency.Sensor.Srange. Start.set and method RsSmab.Sense.Power.Sweep.Frequency.Sensor.Srange.Stop.set.

**param state**
0| 1| OFF| ON

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.1.9 Stop

**SCPI Command :**

```
SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:STOP
```

**class StopCls**

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

**get**(*channel=Channel.Default*) → int

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:STOP
value: int = driver.sense.power.sweep.frequency.sensor.srange.stop.get(channel␣
↪= repcap.Channel.Default)
```

Sets the stop frequency for the frequency power analysis with separate frequencies.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
stop: integer Range: 0 to 1E12

**set**(*stop: int*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:FREQuency:[SENSor]:SRANge:STOP
driver.sense.power.sweep.frequency.sensor.srange.stop.set(stop = 1, channel =␣
↪repcap.Channel.Default)
```

Sets the stop frequency for the frequency power analysis with separate frequencies.

**param stop**
integer Range: 0 to 1E12

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.1.10 Spacing

**SCPI Command :**

```
SENSe:[POWer]:SWEep:FREQuency:SPACing:[MODE]
```

**class SpacingCls**

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

**get_mode**() → MeasRespSpacingMode

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:SPACing:[MODE]
value: enums.MeasRespSpacingMode = driver.sense.power.sweep.frequency.spacing.
↪get_mode()
```

Selects the spacing for the frequency power analysis.

**return**
mode: LINear| LOGarithmic

**set_mode**(*mode: MeasRespSpacingMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:SPACing:[MODE]
driver.sense.power.sweep.frequency.spacing.set_mode(mode = enums.
↪MeasRespSpacingMode.LINear)
```

Selects the spacing for the frequency power analysis.

**param mode**
LINear| LOGarithmic

#### 6.16.1.13.1.11 Timing

##### SCPI Command :

```
SENSe:[POWer]:SWEep:FREQuency:TIMing:[MODE]
```

**class TimingCls**

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

**get_mode**() → MeasRespTimingMode

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:TIMing:[MODE]
value: enums.MeasRespTimingMode = driver.sense.power.sweep.frequency.timing.get_
↪mode()
```

Selects the mode in terms of speed and precision of the response of a measurement.

**return**
mode: FAST| NORMal| HPRecision | FAST| NORMal FAST Selection FAST leads to a fast measurement with a short integration time for each measurement step. NORMal NORMal leads to a longer but more precise measurement due to a higher integration time for each step.

**set_mode**(*mode: MeasRespTimingMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:TIMing:[MODE]
driver.sense.power.sweep.frequency.timing.set_mode(mode = enums.
↪MeasRespTimingMode.FAST)
```

Selects the mode in terms of speed and precision of the response of a measurement.

**param mode**
FAST| NORMal| HPRecision | FAST| NORMal FAST Selection FAST leads to a fast measurement with a short integration time for each measurement step. NORMal NORMal leads to a longer but more precise measurement due to a higher integration time for each step.

#### 6.16.1.13.1.12 Yscale

**SCPI Commands :**

```
SENSe:[POWer]:SWEep:FREQuency:YSCale:MAXimum
SENSe:[POWer]:SWEep:FREQuency:YSCale:MINimum
```

**class YscaleCls**

Yscale commands group definition. 4 total commands, 1 Subgroups, 2 group commands

**get_maximum**() → float

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:MAXimum
value: float = driver.sense.power.sweep.frequency.yscale.get_maximum()
```

Sets the maximum value for the y axis of the measurement diagram.

**return**
maximum: float Range: -200 to 100

**get_minimum**() → float

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:MINimum
value: float = driver.sense.power.sweep.frequency.yscale.get_minimum()
```

Sets the minimum value for the y axis of the measurement diagram.

**return**
minimum: float Range: -200 to 100

**set_maximum**(*maximum: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:MAXimum
driver.sense.power.sweep.frequency.yscale.set_maximum(maximum = 1.0)
```

Sets the maximum value for the y axis of the measurement diagram.

**param maximum**
float Range: -200 to 100

**set_minimum**(*minimum: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:MINimum
driver.sense.power.sweep.frequency.yscale.set_minimum(minimum = 1.0)
```

Sets the minimum value for the y axis of the measurement diagram.

**param minimum**
float Range: -200 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.frequency.yscale.clone()
```

### Subgroups

#### 6.16.1.13.1.13 Auto

### SCPI Commands :

```
SENSe:[POWer]:SWEep:FREQuency:YSCale:AUTO:RESet
SENSe:[POWer]:SWEep:FREQuency:YSCale:AUTO
```

**class AutoCls**

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

**get_value**() → MeasRespYsCaleMode

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:AUTO
value: enums.MeasRespYsCaleMode = driver.sense.power.sweep.frequency.yscale.
↪auto.get_value()
```

Activates autoscaling of the Y axis of the diagram.

**return**

auto: OFF| CEXPanding| FEXPanding| CFLoating| FFLoating OFF Auto scaling is deactivated. If switching from activated to deactivated Auto scaling, the scaling is maintained. CEXPanding | FEXPanding Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is expanded if the minimum or maximum values of the trace move outside the current scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine. CFLoating | FFLoating Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is either expanded if the minimum or maximum values of the trace move outside the current scale or scaled down if the trace fits into a reduced scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine.

**reset**() → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:AUTO:RESet
driver.sense.power.sweep.frequency.yscale.auto.reset()
```

Resets the Y scale to suitable values after the use of auto scaling in the expanding mode. For this mode, the scale might get expanded because of temporarily high-power values. The reset function resets the diagram in such a way that it matches smaller power values again.

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

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:AUTO:RESet
driver.sense.power.sweep.frequency.yscale.auto.reset_with_opc()
```

Resets the Y scale to suitable values after the use of auto scaling in the expanding mode. For this mode, the scale might get expanded because of temporarily high-power values. The reset function resets the diagram in such a way that it matches smaller power values again.

Same as reset, but waits for the operation to complete before continuing further. Use the RsSmab.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**(*auto: MeasRespYsCaleMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:FREQuency:YSCale:AUTO
driver.sense.power.sweep.frequency.yscale.auto.set_value(auto = enums.
↪MeasRespYsCaleMode.CEXPanding)
```

Activates autoscaling of the Y axis of the diagram.

**param auto**
OFF| CEXPanding| FEXPanding| CFLoating| FFLoating OFF Auto scaling is deactivated. If switching from activated to deactivated Auto scaling, the scaling is maintained. CEXPanding | FEXPanding Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is expanded if the minimum or maximum values of the trace move outside the current scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine. CFLoating | FFLoating Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is either expanded if the minimum or maximum values of the trace move outside the current scale or scaled down if the trace fits into a reduced scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine.

#### 6.16.1.13.2 HardCopy

##### SCPI Command :

```
SENSe:[POWer]:SWEep:HCOPy:DATA
```

**class HardCopyCls**

HardCopy commands group definition. 24 total commands, 3 Subgroups, 1 group commands

**get_data**() → bytes

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DATA
value: bytes = driver.sense.power.sweep.hardCopy.get_data()
```

Queries the measurement data directly. The data is transferred to the remote client as data stream. Readable ASCII data is available for hardcopy language CSV. The representation of the values depends on the selected orientation for the CSV format.

**return**
data: block data

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.clone()
```

### Subgroups

#### 6.16.1.13.2.1 Device

### SCPI Commands :

```
SENSe:[POWer]:SWEep:HCOPy:DEVice:SIZE
SENSe:[POWer]:SWEep:HCOPy:DEVice
```

**class DeviceCls**

Device commands group definition. 7 total commands, 1 Subgroups, 2 group commands

**get_size**() → List[int]

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:SIZE
value: List[int] = driver.sense.power.sweep.hardCopy.device.get_size()
```

Sets the size of the hardcopy in number of pixels. The first value of the size setting defines the width, the second value the height of the image.

**return**
size: 320,240 | 640,480 | 800,600 | 1024,768

**get_value**() → HcopyDestination

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice
value: enums.HcopyDestination = driver.sense.power.sweep.hardCopy.device.get_
↪value()
```

Defines the output device. The setting is fixed to FILE, i.e. the hardcopy is stored in a file.

**return**
device: FILE| PRINter

**set_size**(*size: List[int]*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:SIZE
driver.sense.power.sweep.hardCopy.device.set_size(size = [1, 2, 3])
```

Sets the size of the hardcopy in number of pixels. The first value of the size setting defines the width, the second value the height of the image.

**param size**
320,240 | 640,480 | 800,600 | 1024,768

**set_value**(*device: HcopyDestination*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice
driver.sense.power.sweep.hardCopy.device.set_value(device = enums.
↪HcopyDestination.FILE)
```

Defines the output device. The setting is fixed to FILE, i.e. the hardcopy is stored in a file.

**param device**
FILE| PRINter

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.device.clone()
```

### Subgroups

#### 6.16.1.13.2.2 Language

### SCPI Command :

```
SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage
```

**class LanguageCls**

Language commands group definition. 5 total commands, 1 Subgroups, 1 group commands

**get_value**() → MeasRespHcOpFileFormat

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage
value: enums.MeasRespHcOpFileFormat = driver.sense.power.sweep.hardCopy.device.
↪language.get_value()
```

Selects the bitmap graphic format for the screenshot of the power analysis trace. In addition, ASCII file format *.csv is offered. If file format *.csv is selected, the trace data is saved as an ASCII file with comma separated values. It is also possible to directly retrieve the data using command method RsSmab.Sense.Power.Sweep.HardCopy.data

**return**
language: BMP| JPG| XPM| PNG| CSV

**set_value**(*language: MeasRespHcOpFileFormat*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage
driver.sense.power.sweep.hardCopy.device.language.set_value(language = enums.
↪MeasRespHcOpFileFormat.BMP)
```

Selects the bitmap graphic format for the screenshot of the power analysis trace. In addition, ASCII file format *.csv is offered. If file format *.csv is selected, the trace data is saved as an ASCII file with comma separated values. It is also possible to directly retrieve the data using command method RsSmab.Sense.Power.Sweep.HardCopy.data

**param language**
BMP| JPG| XPM| PNG| CSV

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.device.language.clone()
```

### Subgroups

#### 6.16.1.13.2.3 Csv

### SCPI Commands :

```
SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:DPOint
SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:HEADer
SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:ORIentation
```

**class CsvCls**

Csv commands group definition. 4 total commands, 1 Subgroups, 3 group commands

**get_dpoint**() → DecimalSeparator

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:DPOint
value: enums.DecimalSeparator = driver.sense.power.sweep.hardCopy.device.
↪language.csv.get_dpoint()
```

Defines which character is used as the decimal point of the values, either dot or comma.

**return**
dpoint: DOT| COMMa

**get_header**() → MeasRespHcOpCsvhEader

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:HEADer
value: enums.MeasRespHcOpCsvhEader = driver.sense.power.sweep.hardCopy.device.
↪language.csv.get_header()
```

Defines whether each row (or column depending on the orientation) should be preceded by a header containing information about the trace (see also method RsSmab.Sense.Power.Sweep.HardCopy.data) .

**return**
header: OFF| STANdard

**get_orientation**() → MeasRespHcOpCsvoRient

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:ORIentation
value: enums.MeasRespHcOpCsvoRient = driver.sense.power.sweep.hardCopy.device.
↪language.csv.get_orientation()
```

Defines the orientation of the X/Y value pairs.

**return**
orientation: HORizontal| VERTical

**set_dpoint**(*dpoint: DecimalSeparator*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:DPOint
driver.sense.power.sweep.hardCopy.device.language.csv.set_dpoint(dpoint = enums.
↪DecimalSeparator.COMMa)
```

Defines which character is used as the decimal point of the values, either dot or comma.

**param dpoint**
DOT| COMMa

**set_header**(*header: MeasRespHcOpCsvhEader*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:HEADer
driver.sense.power.sweep.hardCopy.device.language.csv.set_header(header = enums.
↪MeasRespHcOpCsvhEader.OFF)
```

Defines whether each row (or column depending on the orientation) should be preceded by a header containing information about the trace (see also method RsSmab.Sense.Power.Sweep.HardCopy.data) .

**param header**
OFF| STANdard

**set_orientation**(*orientation: MeasRespHcOpCsvoRient*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:ORIentation
driver.sense.power.sweep.hardCopy.device.language.csv.set_
↪orientation(orientation = enums.MeasRespHcOpCsvoRient.HORizontal)
```

Defines the orientation of the X/Y value pairs.

**param orientation**
HORizontal| VERTical

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.device.language.csv.clone()
```

### Subgroups

#### 6.16.1.13.2.4 Column

### SCPI Command :

```
SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:[COLumn]:SEParator
```

**class ColumnCls**

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

**get_separator**() → MeasRespHcOpCsvcLmSep

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:[COLumn]:SEParator
value: enums.MeasRespHcOpCsvcLmSep = driver.sense.power.sweep.hardCopy.device.
↪language.csv.column.get_separator()
```

Defines which character is to separate the values, either tabulator, semicolon, comma or blank.

**return**
separator: TABulator| SEMicolon| COMMa| BLANk

**set_separator**(*separator: MeasRespHcOpCsvcLmSep*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:DEVice:LANGuage:CSV:[COLumn]:SEParator
driver.sense.power.sweep.hardCopy.device.language.csv.column.set_
↪separator(separator = enums.MeasRespHcOpCsvcLmSep.BLANk)
```

Defines which character is to separate the values, either tabulator, semicolon, comma or blank.

**param separator**
TABulator| SEMicolon| COMMa| BLANk

#### 6.16.1.13.2.5 Execute

**SCPI Command :**

```
SENSe:[POWer]:SWEep:HCOPy:[EXECute]
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:[EXECute]
driver.sense.power.sweep.hardCopy.execute.set()
```

Triggers the generation of a hardcopy of the current measurement diagram. The data is written into the file selected/created with the method RsSmab.Sense.Power.Sweep.HardCopy.File.Name.value command.

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

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:[EXECute]
driver.sense.power.sweep.hardCopy.execute.set_with_opc()
```

Triggers the generation of a hardcopy of the current measurement diagram. The data is written into the file selected/created with the method RsSmab.Sense.Power.Sweep.HardCopy.File.Name.value command.

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

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

#### 6.16.1.13.2.6 File

**class FileCls**

File commands group definition. 15 total commands, 1 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.file.clone()
```

### Subgroups

#### 6.16.1.13.2.7 Name

### SCPI Command :

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]
```

**class NameCls**

Name commands group definition. 15 total commands, 1 Subgroups, 1 group commands

**get_value**() → str

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]
value: str = driver.sense.power.sweep.hardCopy.file.name.get_value()
```

Creates of selects a file for storing the hardcopy after the method RsSmab.Sense.Power.Sweep.HardCopy.Execute.set command is sent. The directory is either defined with the command method RsSmab.MassMemory.currentDirectory or the path is specified together with the file name. Access to the file via remote control is possible using the commands of the MMEM-Subsystem. In contrast, command method RsSmab.Sense.Power.Sweep.HardCopy.data transfers the hardcopy contents directly to the remote client where they can be further processed.

**return**
name: string

**set_value**(*name: str*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]
driver.sense.power.sweep.hardCopy.file.name.set_value(name = 'abc')
```

Creates of selects a file for storing the hardcopy after the method RsSmab.Sense.Power.Sweep.HardCopy.Execute.set command is sent. The directory is either defined with the command method RsSmab.MassMemory.currentDirectory or the path is specified together with the file name. Access to the file via remote control is possible using the commands of the MMEM-Subsystem. In contrast, command method RsSmab.Sense.Power.Sweep.HardCopy.data transfers the hardcopy contents directly to the remote client where they can be further processed.

**param name**
string

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.file.name.clone()
```

### Subgroups

#### 6.16.1.13.2.8 Auto

### SCPI Commands :

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:STATe
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO
```

**class AutoCls**

Auto commands group definition. 14 total commands, 2 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:STATe
value: bool = driver.sense.power.sweep.hardCopy.file.name.auto.get_state()
```

Activates/deactivates automatic naming of the hardcopy files.

**return**
state: 0| 1| OFF| ON

**get_value**() → str

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO
value: str = driver.sense.power.sweep.hardCopy.file.name.auto.get_value()
```

No command help available

**return**
path_name: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:STATe
driver.sense.power.sweep.hardCopy.file.name.auto.set_state(state = False)
```

Activates/deactivates automatic naming of the hardcopy files.

**param state**
0| 1| OFF| ON

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.file.name.auto.clone()
```

### Subgroups

#### 6.16.1.13.2.9 Directory

### SCPI Commands :

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:DIRectory
```

**class DirectoryCls**

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

**clear**() → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
driver.sense.power.sweep.hardCopy.file.name.auto.directory.clear()
```

Deletes all files with extensions bmp , img, png, xpm and csv in the directory set for automatic naming.

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

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
driver.sense.power.sweep.hardCopy.file.name.auto.directory.clear_with_opc()
```

Deletes all files with extensions bmp , img, png, xpm and csv in the directory set for automatic naming.

Same as clear, but waits for the operation to complete before continuing further. Use the RsSmab.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: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:DIRectory
value: str = driver.sense.power.sweep.hardCopy.file.name.auto.directory.get_
↪value()
```

Defines the directory into which the hardcopy files are stored if auto naming is activated (SENS:SWE:HCOP:FILE:AUTO:STAT ON) .

**return**
directory: string

**set_value**(*directory: str*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:DIRectory
driver.sense.power.sweep.hardCopy.file.name.auto.directory.set_value(directory␣
↪= 'abc')
```

Defines the directory into which the hardcopy files are stored if auto naming is activated (SENS:SWE:HCOP:FILE:AUTO:STAT ON) .

**param directory**
string

#### 6.16.1.13.2.10 File

#### SCPI Commands :

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:NUMBer
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:FILE
```

**class FileCls**

File commands group definition. 10 total commands, 4 Subgroups, 2 group commands

**get_number**() → int

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:NUMBer
value: int = driver.sense.power.sweep.hardCopy.file.name.auto.file.get_number()
```

Queries the generated number in the automatic file name.

**return**
number: integer Range: 0 to 999999

**get_value**() → str

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:FILE
value: str = driver.sense.power.sweep.hardCopy.file.name.auto.file.get_value()
```

Queries the file name generated with the automatic naming settings. Note: As default the automatically generated file name is composed of: >PAth>/<Prefix><YYYY><MM><DD><Number>.<Format>. Each component can be deactivated/ activated separately to individually design the file name.

**return**
file: string

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.hardCopy.file.name.auto.file.clone()
```

#### Subgroups

#### 6.16.1.13.2.11 Day

#### SCPI Commands :

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY
```

**class DayCls**

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

**get_state**() → bool

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
value: bool = driver.sense.power.sweep.hardCopy.file.name.auto.file.day.get_
↪state()
```

Activates the usage of the day in the automatic file name.

**return**
state: 0| 1| OFF| ON

**get_value**() → int

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY
value: int = driver.sense.power.sweep.hardCopy.file.name.auto.file.day.get_
↪value()
```

Queries the day of the date part in the automatic file name.

**return**
day: integer Range: 1 to 31

**set_state**(*state: bool*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
driver.sense.power.sweep.hardCopy.file.name.auto.file.day.set_state(state =␣
↪False)
```

Activates the usage of the day in the automatic file name.

**param state**
0| 1| OFF| ON

#### 6.16.1.13.2.12 Month

**SCPI Commands :**

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh
```

**class MonthCls**

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

**get_state**() → bool

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
value: bool = driver.sense.power.sweep.hardCopy.file.name.auto.file.month.get_
↪state()
```

Activates the usage of the month in the automatic file name.

**return**
state: 0| 1| OFF| ON

**get_value**() → int

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh
value: int = driver.sense.power.sweep.hardCopy.file.name.auto.file.month.get_
↪value()
```

Queries the day of the date part in the automatic file name.

**return**
month: integer Range: 1 to 12

**set_state**(*state: bool*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
driver.sense.power.sweep.hardCopy.file.name.auto.file.month.set_state(state =␣
↪False)
```

Activates the usage of the month in the automatic file name.

**param state**
0| 1| OFF| ON

#### 6.16.1.13.2.13 Prefix

##### SCPI Commands :

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
```

**class PrefixCls**

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

**get_state**() → bool

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
value: bool = driver.sense.power.sweep.hardCopy.file.name.auto.file.prefix.get_
↪state()
```

Activates the usage of the prefix in the automatic file name.

**return**
state: 0| 1| OFF| ON

**get_value**() → str

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
value: str = driver.sense.power.sweep.hardCopy.file.name.auto.file.prefix.get_
↪value()
```

Sets the prefix part in the automatic file name.

**return**
prefix: string

**set_state**(*state: bool*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
driver.sense.power.sweep.hardCopy.file.name.auto.file.prefix.set_state(state =␣
↪False)
```

Activates the usage of the prefix in the automatic file name.

**param state**
0| 1| OFF| ON

**set_value**(*prefix: str*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
driver.sense.power.sweep.hardCopy.file.name.auto.file.prefix.set_value(prefix =
↪'abc')
```

Sets the prefix part in the automatic file name.

**param prefix**
string

#### 6.16.1.13.2.14 Year

**SCPI Commands :**

```
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR
```

**class YearCls**

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

**get_state**() → bool

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
value: bool = driver.sense.power.sweep.hardCopy.file.name.auto.file.year.get_
↪state()
```

Activates the usage of the year in the automatic file name.

**return**
state: 0| 1| OFF| ON

**get_value**() → int

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR
value: int = driver.sense.power.sweep.hardCopy.file.name.auto.file.year.get_
↪value()
```

Queries the year of the date part in the automatic file name.

**return**
year: integer Range: 1784 to 8000

**set_state**(*state: bool*) → None

```
# SCPI: SENSe:[POWer]:SWEep:HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
driver.sense.power.sweep.hardCopy.file.name.auto.file.year.set_state(state =␣
↪False)
```

Activates the usage of the year in the automatic file name.

**param state**
0| 1| OFF| ON

#### 6.16.1.13.3 Power

### SCPI Commands :

```
SENSe:[POWer]:SWEep:POWer:RMODe
SENSe:[POWer]:SWEep:POWer:STARt
SENSe:[POWer]:SWEep:POWer:STEPs
SENSe:[POWer]:SWEep:POWer:STOP
```

**class PowerCls**

Power commands group definition. 18 total commands, 5 Subgroups, 4 group commands

**get_rmode**() → RepeatMode

```
# SCPI: SENSe:[POWer]:SWEep:POWer:RMODe
value: enums.RepeatMode = driver.sense.power.sweep.power.get_rmode()
```

Selects single or continuous mode for measurement mode power in power analysis.

**return**
rmode: SINGle| CONTinuous

**get_start**() → float

```
# SCPI: SENSe:[POWer]:SWEep:POWer:STARt
value: float = driver.sense.power.sweep.power.get_start()
```

Sets the start level for the power versus power measurement.

**return**
start: float Range: -145 to 20

**get_steps**() → int

```
# SCPI: SENSe:[POWer]:SWEep:POWer:STEPs
value: int = driver.sense.power.sweep.power.get_steps()
```

Sets the number of measurement steps for the power versus power measurement.

**return**
steps: integer Range: 10 to 1000

**get_stop**() → float

```
# SCPI: SENSe:[POWer]:SWEep:POWer:STOP
value: float = driver.sense.power.sweep.power.get_stop()
```

Sets the stop level for the power versus power measurement.

**return**
stop: float Range: -145 to 20

**set_rmode**(*rmode: RepeatMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:RMODe
driver.sense.power.sweep.power.set_rmode(rmode = enums.RepeatMode.CONTinuous)
```

Selects single or continuous mode for measurement mode power in power analysis.

**param rmode**
SINGle| CONTinuous

**set_start**(*start: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:STARt
driver.sense.power.sweep.power.set_start(start = 1.0)
```

Sets the start level for the power versus power measurement.

**param start**
float Range: -145 to 20

**set_steps**(*steps: int*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:STEPs
driver.sense.power.sweep.power.set_steps(steps = 1)
```

Sets the number of measurement steps for the power versus power measurement.

**param steps**
integer Range: 10 to 1000

**set_stop**(*stop: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:STOP
driver.sense.power.sweep.power.set_stop(stop = 1.0)
```

Sets the stop level for the power versus power measurement.

**param stop**
float Range: -145 to 20

### Cloning the Group

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

### Subgroups

#### 6.16.1.13.3.1 Reference

**class ReferenceCls**

Reference 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.sense.power.sweep.power.reference.clone()
```

### Subgroups

#### 6.16.1.13.3.2 Data

### SCPI Commands :

```
SENSe:[POWer]:SWEep:POWer:REFerence:DATA:COPY
SENSe:[POWer]:SWEep:POWer:REFerence:DATA:POINts
SENSe:[POWer]:SWEep:POWer:REFerence:DATA:XVALues
SENSe:[POWer]:SWEep:POWer:REFerence:DATA:YVALues
```

**class DataCls**

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

**copy**() → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:REFerence:DATA:COPY
driver.sense.power.sweep.power.reference.data.copy()
```

Generates a reference curve for 'Power' measurement.

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

```
# SCPI: SENSe:[POWer]:SWEep:POWer:REFerence:DATA:COPY
driver.sense.power.sweep.power.reference.data.copy_with_opc()
```

Generates a reference curve for 'Power' measurement.

Same as copy, but waits for the operation to complete before continuing further. Use the RsSmab.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_points**() → int

```
# SCPI: SENSe:[POWer]:SWEep:POWer:REFerence:DATA:POINts
value: int = driver.sense.power.sweep.power.reference.data.get_points()
```

Queries the number of points from the reference curve in 'Power' measurement.

**return**
points: integer Range: 10 to 1000

**get_xvalues**() → List[float]

```
# SCPI: SENSe:[POWer]:SWEep:POWer:REFerence:DATA:XVALues
value: List[float] = driver.sense.power.sweep.power.reference.data.get_xvalues()
```

Sets or queries the x values of the two reference points, i.e. 'Power X (Point A) ' and 'Power X (Point B) ' in 'Power' measurement.

**return**
xvalues: string

**get_yvalues**() → List[float]

```
# SCPI: SENSe:[POWer]:SWEep:POWer:REFerence:DATA:YVALues
value: List[float] = driver.sense.power.sweep.power.reference.data.get_yvalues()
```

Sets or queries the y values of the two reference points, i.e. 'Power Y (Point A) ' and 'Power Y (Point B) ' in 'Power' measurement.

**return**
yvalues: string

**set_xvalues**(*xvalues: List[float]*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:REFerence:DATA:XVALues
driver.sense.power.sweep.power.reference.data.set_xvalues(xvalues = [1.1, 2.2,␣
↪3.3])
```

Sets or queries the x values of the two reference points, i.e. 'Power X (Point A) ' and 'Power X (Point B) ' in 'Power' measurement.

**param xvalues**
string

**set_yvalues**(*yvalues: List[float]*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:REFerence:DATA:YVALues
driver.sense.power.sweep.power.reference.data.set_yvalues(yvalues = [1.1, 2.2,␣
↪3.3])
```

Sets or queries the y values of the two reference points, i.e. 'Power Y (Point A) ' and 'Power Y (Point B) ' in 'Power' measurement.

**param yvalues**
string

#### 6.16.1.13.3.3 Sensor

**class SensorCls**

Sensor 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.sense.power.sweep.power.sensor.clone()
```

### Subgroups

#### 6.16.1.13.3.4 Offset

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:OFFSet
```

**class OffsetCls**

Offset commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:OFFSet
value: float = driver.sense.power.sweep.power.sensor.offset.get(channel =␣
↪repcap.Channel.Default)
```

Defines the level offset at the sensor input in dB. Activate the offset with the command method RsSmab.Sense.Power.Sweep. Power.Sensor.Offset.State.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
offset: float Range: -100 to 100

**set**(*offset: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:OFFSet
driver.sense.power.sweep.power.sensor.offset.set(offset = 1.0, channel = repcap.
↪Channel.Default)
```

Defines the level offset at the sensor input in dB. Activate the offset with the command method RsSmab.Sense.Power.Sweep. Power.Sensor.Offset.State.set.

**param offset**
float Range: -100 to 100

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.power.sensor.offset.clone()
```

### Subgroups

#### 6.16.1.13.3.5 State

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:OFFSet:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:OFFSet:STATe
value: bool = driver.sense.power.sweep.power.sensor.offset.state.get(channel =␣
↪repcap.Channel.Default)
```

Activates a level offset at the sensor input. Define the appropriate value with the command method RsSmab.Sense.Power. Sweep.Power.Sensor.Offset.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:OFFSet:STATe
driver.sense.power.sweep.power.sensor.offset.state.set(state = False, channel =␣
↪repcap.Channel.Default)
```

Activates a level offset at the sensor input. Define the appropriate value with the command method RsSmab.Sense.Power. Sweep.Power.Sensor.Offset.set.

**param state**
0| 1| OFF| ON

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.3.6 Sfrequency

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:SFRequency
```

**class SfrequencyCls**

Sfrequency commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:SFRequency
value: float = driver.sense.power.sweep.power.sensor.sfrequency.get(channel =␣
↪repcap.Channel.Default)
```

Defines the separate frequency used for power vs. power measurement.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
sfrequency: float Range: 0 to 1E12

**set**(*sfrequency: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:SFRequency
driver.sense.power.sweep.power.sensor.sfrequency.set(sfrequency = 1.0, channel␣
↪= repcap.Channel.Default)
```

Defines the separate frequency used for power vs. power measurement.

**param sfrequency**
float Range: 0 to 1E12

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.power.sensor.sfrequency.clone()
```

### Subgroups

#### 6.16.1.13.3.7 State

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:SFRequency:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:SFRequency:STATe
value: bool = driver.sense.power.sweep.power.sensor.sfrequency.state.
↪get(channel = repcap.Channel.Default)
```

Activates the use of a different frequency for the power measurement.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:POWer:[SENSor]:SFRequency:STATe
driver.sense.power.sweep.power.sensor.sfrequency.state.set(state = False,␣
↪channel = repcap.Channel.Default)
```

Activates the use of a different frequency for the power measurement.

**param state**
0| 1| OFF| ON

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.3.8 Spacing

**SCPI Command :**

```
SENSe:[POWer]:SWEep:POWer:SPACing:[MODE]
```

**class SpacingCls**

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

**get_mode**() → MeasRespSpacingMode

```
# SCPI: SENSe:[POWer]:SWEep:POWer:SPACing:[MODE]
value: enums.MeasRespSpacingMode = driver.sense.power.sweep.power.spacing.get_
↪mode()
```

Selects the spacing for the frequency power analysis.

**return**
mode: LINear

**set_mode**(*mode: MeasRespSpacingMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:SPACing:[MODE]
driver.sense.power.sweep.power.spacing.set_mode(mode = enums.
↪MeasRespSpacingMode.LINear)
```

Selects the spacing for the frequency power analysis.

**param mode**
LINear

#### 6.16.1.13.3.9 Timing

**SCPI Command :**

```
SENSe:[POWer]:SWEep:POWer:TIMing:[MODE]
```

**class TimingCls**

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

**get_mode**() → MeasRespTimingMode

```
# SCPI: SENSe:[POWer]:SWEep:POWer:TIMing:[MODE]
value: enums.MeasRespTimingMode = driver.sense.power.sweep.power.timing.get_
↪mode()
```

Selects the timing mode of the measurement.

**return**
mode: FAST| NORMal| HPRecision | FAST| NORMal FAST Selection FAST leads to a fast measurement with a short integration times for each measurement step. NORMal NORMal leads to a longer but more precise measurement due to a higher integration time for each step.

**set_mode**(*mode: MeasRespTimingMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:TIMing:[MODE]
driver.sense.power.sweep.power.timing.set_mode(mode = enums.MeasRespTimingMode.
↪FAST)
```

Selects the timing mode of the measurement.

**param mode**
FAST| NORMal| HPRecision | FAST| NORMal FAST Selection FAST leads to a fast measurement with a short integration times for each measurement step. NORMal NORMal leads to a longer but more precise measurement due to a higher integration time for each step.

#### 6.16.1.13.3.10 Yscale

**SCPI Commands :**

```
SENSe:[POWer]:SWEep:POWer:YSCale:MAXimum
SENSe:[POWer]:SWEep:POWer:YSCale:MINimum
```

**class YscaleCls**

Yscale commands group definition. 4 total commands, 1 Subgroups, 2 group commands

**get_maximum**() → float

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:MAXimum
value: float = driver.sense.power.sweep.power.yscale.get_maximum()
```

Sets the maximum value for the y axis of the measurement diagram.

**return**
maximum: float Range: -200 to 100

**get_minimum**() → float

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:MINimum
value: float = driver.sense.power.sweep.power.yscale.get_minimum()
```

Sets the minimum value for the y axis of the measurement diagram.

**return**
minimum: float Range: -200 to 100

**set_maximum**(*maximum: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:MAXimum
driver.sense.power.sweep.power.yscale.set_maximum(maximum = 1.0)
```

Sets the maximum value for the y axis of the measurement diagram.

**param maximum**
float Range: -200 to 100

**set_minimum**(*minimum: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:MINimum
driver.sense.power.sweep.power.yscale.set_minimum(minimum = 1.0)
```

Sets the minimum value for the y axis of the measurement diagram.

**param minimum**
float Range: -200 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.power.yscale.clone()
```

### Subgroups

#### 6.16.1.13.3.11 Auto

### SCPI Commands :

```
SENSe:[POWer]:SWEep:POWer:YSCale:AUTO:RESet
SENSe:[POWer]:SWEep:POWer:YSCale:AUTO
```

**class AutoCls**

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

**get_value**() → MeasRespYsCaleMode

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:AUTO
value: enums.MeasRespYsCaleMode = driver.sense.power.sweep.power.yscale.auto.
↪get_value()
```

Activates autoscaling of the Y axis of the diagram.

**return**

auto: OFF| CEXPanding| FEXPanding| CFLoating| FFLoating OFF Auto scaling is deactivated. When switching from activated to deactivated Auto scaling, the scaling is maintained. When switching from deactivated to activated Auto scaling, the scaling is reset to min = max = 0. CEXPanding | FEXPanding Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is expanded if the minimum or maximum values of the trace move outside the current scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine. CFLoating | FFLoating Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is either expanded if the minimum or maximum values of the trace move outside the current scale or scaled down if the trace fits into a reduced scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine.

**reset**() → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:AUTO:RESet
driver.sense.power.sweep.power.yscale.auto.reset()
```

Resets the Y scale to suitable values after the use of auto scaling in the expanding mode. For this mode, the scale might get expanded because of temporarily high power values. The reset function allows resetting the diagram to match smaller power values again.

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

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:AUTO:RESet
driver.sense.power.sweep.power.yscale.auto.reset_with_opc()
```

Resets the Y scale to suitable values after the use of auto scaling in the expanding mode. For this mode, the scale might get expanded because of temporarily high power values. The reset function allows resetting the diagram to match smaller power values again.

Same as reset, but waits for the operation to complete before continuing further. Use the RsSmab.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**(*auto: MeasRespYsCaleMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:POWer:YSCale:AUTO
driver.sense.power.sweep.power.yscale.auto.set_value(auto = enums.
↪MeasRespYsCaleMode.CEXPanding)
```

Activates autoscaling of the Y axis of the diagram.

**param auto**
OFF| CEXPanding| FEXPanding| CFLoating| FFLoating OFF Auto scaling is deactivated. When switching from activated to deactivated Auto scaling, the scaling is maintained. When switching from deactivated to activated Auto scaling, the scaling is reset to min = max = 0. CEXPanding | FEXPanding Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is expanded if the minimum or maximum values of the trace move outside the current scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine. CFLoating | FFLoating Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is either expanded if the minimum or maximum values of the trace move outside the current scale or scaled down if the trace fits into a reduced scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine.

#### 6.16.1.13.4 Time

**SCPI Commands :**

```
SENSe:[POWer]:SWEep:TIME:RMODe
SENSe:[POWer]:SWEep:TIME:STARt
SENSe:[POWer]:SWEep:TIME:STEPs
SENSe:[POWer]:SWEep:TIME:STOP
SENSe:[POWer]:SWEep:TIME:TEVents
```

**class TimeCls**

Time commands group definition. 30 total commands, 5 Subgroups, 5 group commands

**get_rmode**() → RepeatMode

```
# SCPI: SENSe:[POWer]:SWEep:TIME:RMODe
value: enums.RepeatMode = driver.sense.power.sweep.time.get_rmode()
```

Selects single or continuous mode for measurement mode time in power analysis.

**return**
rmode: SINGle| CONTinuous

**get_start**() → float

```
# SCPI: SENSe:[POWer]:SWEep:TIME:STARt
value: float = driver.sense.power.sweep.time.get_start()
```

Sets the start time for the power versus time measurement. Value 0 defines the trigger point. By choosing a negative time value, the trace can be shifted in the diagram. It is possible, that the measurement cannot be performed over the complete time range because of limitations due to sensor settings. In this case, an error message is output.

**return**
start: float Range: -1 to 1

**get_steps**() → int

```
# SCPI: SENSe:[POWer]:SWEep:TIME:STEPs
value: int = driver.sense.power.sweep.time.get_steps()
```

Sets the number of measurement steps for the power versus time measurement. Value 0 defines the trigger point.

**return**
steps: integer Range: 10 to 1000

**get_stop**() → float

```
# SCPI: SENSe:[POWer]:SWEep:TIME:STOP
value: float = driver.sense.power.sweep.time.get_stop()
```

Sets the stop time for the power versus time measurement.

**return**
stop: float Range: 0 to 2

**get_tevents**() → MeasRespYsCaleEvents

```
# SCPI: SENSe:[POWer]:SWEep:TIME:TEVents
value: enums.MeasRespYsCaleEvents = driver.sense.power.sweep.time.get_tevents()
```

Determines, whether the measurement data processing starts with a trigger event in one of the sensors (Logical OR) , or whether all channels have to be triggered (logical AND) . Each sensor evaluates a trigger event according to its setting independently. This function supports the internal or external trigger modes with multi-channel time measurements.

**return**
trigger_tevents: AND| OR

**set_rmode**(*rmode: RepeatMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:RMODe
driver.sense.power.sweep.time.set_rmode(rmode = enums.RepeatMode.CONTinuous)
```

Selects single or continuous mode for measurement mode time in power analysis.

**param rmode**
SINGle| CONTinuous

**set_start**(*start: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:STARt
driver.sense.power.sweep.time.set_start(start = 1.0)
```

Sets the start time for the power versus time measurement. Value 0 defines the trigger point. By choosing a negative time value, the trace can be shifted in the diagram. It is possible, that the measurement cannot be performed over the complete time range because of limitations due to sensor settings. In this case, an error message is output.

**param start**
float Range: -1 to 1

**set_steps**(*steps: int*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:STEPs
driver.sense.power.sweep.time.set_steps(steps = 1)
```

Sets the number of measurement steps for the power versus time measurement. Value 0 defines the trigger point.

**param steps**
integer Range: 10 to 1000

**set_stop**(*stop: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:STOP
driver.sense.power.sweep.time.set_stop(stop = 1.0)
```

Sets the stop time for the power versus time measurement.

**param stop**
float Range: 0 to 2

**set_tevents**(*trigger_tevents: MeasRespYsCaleEvents*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:TEVents
driver.sense.power.sweep.time.set_tevents(trigger_tevents = enums.
↪MeasRespYsCaleEvents.AND)
```

Determines, whether the measurement data processing starts with a trigger event in one of the sensors (Logical OR) , or whether all channels have to be triggered (logical AND) . Each sensor evaluates a trigger event according to its setting independently. This function supports the internal or external trigger modes with multi-channel time measurements.

**param trigger_tevents**
AND| OR

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.clone()
```

### Subgroups

#### 6.16.1.13.4.1 Average

### SCPI Command :

```
SENSe:[POWer]:SWEep:TIME:AVERage:[COUNt]
```

**class AverageCls**

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

**get_count**() → MeasRespTimeAverage

```
# SCPI: SENSe:[POWer]:SWEep:TIME:AVERage:[COUNt]
value: enums.MeasRespTimeAverage = driver.sense.power.sweep.time.average.get_
↪count()
```

Selects the averaging factor in time mode. The count number determines how many measurement cycles are used to form a measurement result. Higher averaging counts reduce noise but increase the measurement time. Averaging requires a stable trigger event so that the measurement cycles have the same timing.

**return**
count: 1| 2| 4| 8| 16| 32| 64| 128| 256| 512| 1024

**set_count**(*count: MeasRespTimeAverage*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:AVERage:[COUNt]
driver.sense.power.sweep.time.average.set_count(count = enums.
↪MeasRespTimeAverage._1)
```

Selects the averaging factor in time mode. The count number determines how many measurement cycles are used to form a measurement result. Higher averaging counts reduce noise but increase the measurement time. Averaging requires a stable trigger event so that the measurement cycles have the same timing.

**param count**
1| 2| 4| 8| 16| 32| 64| 128| 256| 512| 1024

#### 6.16.1.13.4.2 Reference

**class ReferenceCls**

Reference 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.sense.power.sweep.time.reference.clone()
```

#### Subgroups

##### 6.16.1.13.4.3 Data

#### SCPI Commands :

```
SENSe:[POWer]:SWEep:TIME:REFerence:DATA:COPY
SENSe:[POWer]:SWEep:TIME:REFerence:DATA:POINts
SENSe:[POWer]:SWEep:TIME:REFerence:DATA:XVALues
SENSe:[POWer]:SWEep:TIME:REFerence:DATA:YVALues
```

**class DataCls**

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

**copy**() → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:REFerence:DATA:COPY
driver.sense.power.sweep.time.reference.data.copy()
```

Generates a reference curve for 'Time' measurement.

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

```
# SCPI: SENSe:[POWer]:SWEep:TIME:REFerence:DATA:COPY
driver.sense.power.sweep.time.reference.data.copy_with_opc()
```

Generates a reference curve for 'Time' measurement.

Same as copy, but waits for the operation to complete before continuing further. Use the RsSmab.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_points**() → int

```
# SCPI: SENSe:[POWer]:SWEep:TIME:REFerence:DATA:POINts
value: int = driver.sense.power.sweep.time.reference.data.get_points()
```

Queries the number of points from the reference curve in 'Time' measurement.

**return**
points: integer Range: 10 to 1000

**get_xvalues**() → List[float]

```
# SCPI: SENSe:[POWer]:SWEep:TIME:REFerence:DATA:XVALues
value: List[float] = driver.sense.power.sweep.time.reference.data.get_xvalues()
```

Sets or queries the x values of the two reference points, i.e. 'Time X (Point A) ' and 'Time X (Point B) ' in 'Time' measurement.

**return**
xvalues: string

**get_yvalues**() → List[float]

```
# SCPI: SENSe:[POWer]:SWEep:TIME:REFerence:DATA:YVALues
value: List[float] = driver.sense.power.sweep.time.reference.data.get_yvalues()
```

Sets or queries the y values of the two reference points, i.e. 'Power Y (Point A) ' and 'Power Y (Point B) ' in 'Time' measurement.

**return**
yvalues: string

**set_xvalues**(*xvalues: List[float]*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:REFerence:DATA:XVALues
driver.sense.power.sweep.time.reference.data.set_xvalues(xvalues = [1.1, 2.2, 3.
↪3])
```

Sets or queries the x values of the two reference points, i.e. 'Time X (Point A) ' and 'Time X (Point B) ' in 'Time' measurement.

**param xvalues**
string

**set_yvalues**(*yvalues: List[float]*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:REFerence:DATA:YVALues
driver.sense.power.sweep.time.reference.data.set_yvalues(yvalues = [1.1, 2.2, 3.
↪3])
```

Sets or queries the y values of the two reference points, i.e. 'Power Y (Point A) ' and 'Power Y (Point B) ' in 'Time' measurement.

**param yvalues**
string

#### 6.16.1.13.4.4 Sensor

**class SensorCls**

Sensor commands group definition. 15 total commands, 4 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.sensor.clone()
```

### Subgroups

#### 6.16.1.13.4.5 Offset

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:OFFSet
```

**class OffsetCls**

Offset commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:OFFSet
value: float = driver.sense.power.sweep.time.sensor.offset.get(channel = repcap.
↪Channel.Default)
```

Defines the level offset at the sensor input in dB. Activate the offset with the command method RsSmab.Sense.Power.Sweep. Time.Sensor.Offset.State.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
offset: float Range: -100 to 100

**set**(*offset: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:OFFSet
driver.sense.power.sweep.time.sensor.offset.set(offset = 1.0, channel = repcap.
↪Channel.Default)
```

Defines the level offset at the sensor input in dB. Activate the offset with the command method RsSmab.Sense.Power.Sweep. Time.Sensor.Offset.State.set.

**param offset**
float Range: -100 to 100

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.sensor.offset.clone()
```

### Subgroups

#### 6.16.1.13.4.6 State

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:OFFSet:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:OFFSet:STATe
value: bool = driver.sense.power.sweep.time.sensor.offset.state.get(channel =␣
↪repcap.Channel.Default)
```

Activates a level offset at the sensor input. Define the appropriate value with the command method RsSmab.Sense.Power. Sweep.Time.Sensor.Offset.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:OFFSet:STATe
driver.sense.power.sweep.time.sensor.offset.state.set(state = False, channel =␣
↪repcap.Channel.Default)
```

Activates a level offset at the sensor input. Define the appropriate value with the command method RsSmab.Sense.Power. Sweep.Time.Sensor.Offset.set.

**param state**
0| 1| OFF| ON

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.7 Pulse

**class PulseCls**

Pulse commands group definition. 5 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.sensor.pulse.clone()
```

### Subgroups

#### 6.16.1.13.4.8 State

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:STATe
value: bool = driver.sense.power.sweep.time.sensor.pulse.state.get(channel =␣
↪repcap.Channel.Default)
```

Enables pulse data analysis. The measurement is started with command INITiate. Note: The command is only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:STATe
driver.sense.power.sweep.time.sensor.pulse.state.set(state = False, channel =␣
↪repcap.Channel.Default)
```

Enables pulse data analysis. The measurement is started with command INITiate. Note: The command is only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.9 Threshold

**class ThresholdCls**

Threshold 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.sense.power.sweep.time.sensor.pulse.threshold.clone()
```

### Subgroups

#### 6.16.1.13.4.10 Base

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:BASE
```

**class BaseCls**

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

**get**(*channel=Channel.Default*) → MeasRespPulsThrBase

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:BASE
value: enums.MeasRespPulsThrBase = driver.sense.power.sweep.time.sensor.pulse.
↪threshold.base.get(channel = repcap.Channel.Default)
```

Selects how the threshold parameters for pulse analysis are calculated. Note: The command is only available in time measurement mode and with R&S NRPZ81 power sensors.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
base: VOLTage| POWer

**set**(*base: MeasRespPulsThrBase*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:BASE
driver.sense.power.sweep.time.sensor.pulse.threshold.base.set(base = enums.
↪MeasRespPulsThrBase.POWer, channel = repcap.Channel.Default)
```

Selects how the threshold parameters for pulse analysis are calculated. Note: The command is only available in time measurement mode and with R&S NRPZ81 power sensors.

**param base**
VOLTage| POWer

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.11 Power

**class PowerCls**

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

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.sensor.pulse.threshold.power.clone()
```

### Subgroups

#### 6.16.1.13.4.12 Hreference

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:HREFerence
```

**class HreferenceCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:HREFerence
value: float = driver.sense.power.sweep.time.sensor.pulse.threshold.power.
↪hreference.get(channel = repcap.Channel.Default)
```

Sets the upper reference level in terms of percentage of the overall pulse level (power or voltage) . The distal power defines the end of the rising edge and the start of the falling edge of the pulse. Note: The command is only available in time measurement mode and with R&S NRPZ81 power sensors.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
hreference: float Range: 0 to 100

**set**(*hreference: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:HREFerence
driver.sense.power.sweep.time.sensor.pulse.threshold.power.hreference.
↪set(hreference = 1.0, channel = repcap.Channel.Default)
```

Sets the upper reference level in terms of percentage of the overall pulse level (power or voltage) . The distal power defines the end of the rising edge and the start of the falling edge of the pulse. Note: The command is only available in time measurement mode and with R&S NRPZ81 power sensors.

**param hreference**
float Range: 0 to 100

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.13 Lreference

**SCPI Command :**

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:LREFerence
```

**class LreferenceCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:LREFerence
value: float = driver.sense.power.sweep.time.sensor.pulse.threshold.power.
↪lreference.get(channel = repcap.Channel.Default)
```

Sets the lower reference level in terms of percentage of the overall pulse level. The proximal power defines the start of the rising edge and the end of the falling edge of the pulse. Note: This parameter is only available in time measurement mode and R&S NRP-Z81 power sensors.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
lreference: float Range: 0.0 to 100.0

**set**(*lreference: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:LREFerence
driver.sense.power.sweep.time.sensor.pulse.threshold.power.lreference.
↪set(lreference = 1.0, channel = repcap.Channel.Default)
```

Sets the lower reference level in terms of percentage of the overall pulse level. The proximal power defines the start of the rising edge and the end of the falling edge of the pulse. Note: This parameter is only available in time measurement mode and R&S NRP-Z81 power sensors.

**param lreference**
float Range: 0.0 to 100.0

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.14 Reference

**SCPI Command :**

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:REFerence
```

**class ReferenceCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:REFerence
value: float = driver.sense.power.sweep.time.sensor.pulse.threshold.power.
↪reference.get(channel = repcap.Channel.Default)
```

Sets the medial reference level in terms of percentage of the overall pulse level (power or voltage related) . This level is used to define pulse width and pulse period. Note: The command is only available in time measurement mode and with R&S NRPZ81 power sensors.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
reference: float Range: 0.0 to 100.0

**set**(*reference: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:PULSe:THReshold:POWer:REFerence
driver.sense.power.sweep.time.sensor.pulse.threshold.power.reference.
↪set(reference = 1.0, channel = repcap.Channel.Default)
```

Sets the medial reference level in terms of percentage of the overall pulse level (power or voltage related) . This level is used to define pulse width and pulse period. Note: The command is only available in time measurement mode and with R&S NRPZ81 power sensors.

**param reference**
float Range: 0.0 to 100.0

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.15 Sfrequency

#### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:SFRequency
```

**class SfrequencyCls**

Sfrequency commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:SFRequency
value: float = driver.sense.power.sweep.time.sensor.sfrequency.get(channel =␣
↪repcap.Channel.Default)
```

Defines the separate frequency used for power vs. time measurement.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
sfrequency: float Range: 0 to 1E12

**set**(*sfrequency: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:SFRequency
driver.sense.power.sweep.time.sensor.sfrequency.set(sfrequency = 1.0, channel =␣
↪repcap.Channel.Default)
```

Defines the separate frequency used for power vs. time measurement.

**param sfrequency**
float Range: 0 to 1E12

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.sensor.sfrequency.clone()
```

### Subgroups

#### 6.16.1.13.4.16 State

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:SFRequency:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:SFRequency:STATe
value: bool = driver.sense.power.sweep.time.sensor.sfrequency.state.get(channel␣
↪= repcap.Channel.Default)
```

Activates the use of a different frequency for the power measurement.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:SFRequency:STATe
driver.sense.power.sweep.time.sensor.sfrequency.state.set(state = False,␣
↪channel = repcap.Channel.Default)
```

Activates the use of a different frequency for the power measurement.

**param state**
0| 1| OFF| ON

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.17 Trigger

**class TriggerCls**

Trigger commands group definition. 6 total commands, 6 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.sensor.trigger.clone()
```

### Subgroups

#### 6.16.1.13.4.18 Auto

### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:AUTO
```

**class AutoCls**

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

**get**(*channel=Channel.Default*) → MeasRespTrigAutoSet

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:AUTO
value: enums.MeasRespTrigAutoSet = driver.sense.power.sweep.time.sensor.trigger.
↪auto.get(channel = repcap.Channel.Default)
```

Sets the trigger level, the hysteresis and the dropout time to default values.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
auto: ONCE

**set**(*auto: MeasRespTrigAutoSet*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:AUTO
driver.sense.power.sweep.time.sensor.trigger.auto.set(auto = enums.
↪MeasRespTrigAutoSet.ONCE, channel = repcap.Channel.Default)
```

Sets the trigger level, the hysteresis and the dropout time to default values.

**param auto**
ONCE

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.19 Dtime

**SCPI Command :**

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:DTIMe
```

**class DtimeCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:DTIMe
value: float = driver.sense.power.sweep.time.sensor.trigger.dtime.get(channel =␣
↪repcap.Channel.Default)
```

Determines the minimum time for which the signal must be below (above) the power level defined by level and hysteresis before triggering can occur again.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
dtime: float Range: 0 to 10

**set**(*dtime: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:DTIMe
driver.sense.power.sweep.time.sensor.trigger.dtime.set(dtime = 1.0, channel =␣
↪repcap.Channel.Default)
```

Determines the minimum time for which the signal must be below (above) the power level defined by level and hysteresis before triggering can occur again.

**param dtime**
float Range: 0 to 10

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.20 Hysteresis

**SCPI Command :**

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:HYSTeresis
```

**class HysteresisCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:HYSTeresis
value: float = driver.sense.power.sweep.time.sensor.trigger.hysteresis.
↪get(channel = repcap.Channel.Default)
```

Sets the hysteresis of the internal trigger threshold. Hysteresis is the magnitude (in dB) the trigger signal level must drop below the trigger threshold (positive trigger slope) before triggering can occur again.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
hysteresis: float Range: 0 to 10

**set**(*hysteresis: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:HYSTeresis
driver.sense.power.sweep.time.sensor.trigger.hysteresis.set(hysteresis = 1.0,␣
↪channel = repcap.Channel.Default)
```

Sets the hysteresis of the internal trigger threshold. Hysteresis is the magnitude (in dB) the trigger signal level must drop below the trigger threshold (positive trigger slope) before triggering can occur again.

**param hysteresis**
float Range: 0 to 10

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.21 Level

**SCPI Command :**

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:LEVel
```

**class LevelCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:LEVel
value: float = driver.sense.power.sweep.time.sensor.trigger.level.get(channel =␣
↪repcap.Channel.Default)
```

Sets the trigger threshold.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
level: float Range: -200 to 100

**set**(*level: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:LEVel
driver.sense.power.sweep.time.sensor.trigger.level.set(level = 1.0, channel =␣
↪repcap.Channel.Default)
```

Sets the trigger threshold.

**param level**
float Range: -200 to 100

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.22 Slope

#### SCPI Command :

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:SLOPe
```

**class SlopeCls**

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

**get**(*channel=Channel.Default*) → SlopeType

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:SLOPe
value: enums.SlopeType = driver.sense.power.sweep.time.sensor.trigger.slope.
↪get(channel = repcap.Channel.Default)
```

Sets the polarity of the active slope for the trigger signals.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
trigger_slope: POSitive| NEGative

**set**(*trigger_slope: SlopeType*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:SLOPe
driver.sense.power.sweep.time.sensor.trigger.slope.set(trigger_slope = enums.
↪SlopeType.NEGative, channel = repcap.Channel.Default)
```

Sets the polarity of the active slope for the trigger signals.

**param trigger_slope**
POSitive| NEGative

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.23 Source

**SCPI Command :**

```
SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:SOURce
```

**class SourceCls**

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

**get**(*channel=Channel.Default*) → MeasRespTrigMode

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:SOURce
value: enums.MeasRespTrigMode = driver.sense.power.sweep.time.sensor.trigger.
↪source.get(channel = repcap.Channel.Default)
```

Selects if the measurement is free running (FREE) or starts only after a trigger event. The trigger can be applied internally or externally.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
source: FREE| AUTO| INTernal| EXTernal

**set**(*source: MeasRespTrigMode*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SWEep:TIME:[SENSor]:TRIGger:SOURce
driver.sense.power.sweep.time.sensor.trigger.source.set(source = enums.
↪MeasRespTrigMode.AUTO, channel = repcap.Channel.Default)
```

Selects if the measurement is free running (FREE) or starts only after a trigger event. The trigger can be applied internally or externally.

**param source**
FREE| AUTO| INTernal| EXTernal

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.16.1.13.4.24 Spacing

**SCPI Command :**

```
SENSe:[POWer]:SWEep:TIME:SPACing:[MODE]
```

**class SpacingCls**

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

**get_mode**() → MeasRespSpacingMode

```
# SCPI: SENSe:[POWer]:SWEep:TIME:SPACing:[MODE]
value: enums.MeasRespSpacingMode = driver.sense.power.sweep.time.spacing.get_
↪mode()
```

Queries the sweep spacing for the power versus time measurement. The spacing is fixed to linear.

**return**
mode: LINear

**set_mode**(*mode: MeasRespSpacingMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:SPACing:[MODE]
driver.sense.power.sweep.time.spacing.set_mode(mode = enums.MeasRespSpacingMode.
↪LINear)
```

Queries the sweep spacing for the power versus time measurement. The spacing is fixed to linear.

**param mode**
LINear

#### 6.16.1.13.4.25 Yscale

#### SCPI Commands :

```
SENSe:[POWer]:SWEep:TIME:YSCale:MAXimum
SENSe:[POWer]:SWEep:TIME:YSCale:MINimum
```

**class YscaleCls**

Yscale commands group definition. 4 total commands, 1 Subgroups, 2 group commands

**get_maximum**() → float

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:MAXimum
value: float = driver.sense.power.sweep.time.yscale.get_maximum()
```

Sets the maximum value for the y axis of the measurement diagram.

**return**
maximum: float Range: -200 to 100, Unit: dBm

**get_minimum**() → float

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:MINimum
value: float = driver.sense.power.sweep.time.yscale.get_minimum()
```

Sets the minimum value for the y axis of the measurement diagram.

**return**
minimum: float Range: -200 to 100

**set_maximum**(*maximum: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:MAXimum
driver.sense.power.sweep.time.yscale.set_maximum(maximum = 1.0)
```

Sets the maximum value for the y axis of the measurement diagram.

**param maximum**
float Range: -200 to 100, Unit: dBm

**set_minimum**(*minimum: float*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:MINimum
driver.sense.power.sweep.time.yscale.set_minimum(minimum = 1.0)
```

Sets the minimum value for the y axis of the measurement diagram.

**param minimum**
float Range: -200 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.power.sweep.time.yscale.clone()
```

### Subgroups

#### 6.16.1.13.4.26 Auto

### SCPI Commands :

```
SENSe:[POWer]:SWEep:TIME:YSCale:AUTO:RESet
SENSe:[POWer]:SWEep:TIME:YSCale:AUTO
```

**class AutoCls**

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

**get_value**() → MeasRespYsCaleMode

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:AUTO
value: enums.MeasRespYsCaleMode = driver.sense.power.sweep.time.yscale.auto.get_
↪value()
```

Activates autoscaling of the Y axis in the diagram.

**return**
auto: OFF| CEXPanding| FEXPanding| CFLoating| FFLoating OFF Auto scaling is deactivated. When switching from activated to deactivated Auto scaling, the scaling is maintained. When switching from deactivated to activated Auto scaling, the scaling is reset to min = max = 0. CEXPanding | FEXPanding Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is expanded if the minimum or maximum values of the trace move outside the current scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine. CFLoating | FFLoating Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is either expanded if the minimum or maximum values of the trace move outside the current scale or scaled down if the trace fits into a reduced scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine.

**reset**() → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:AUTO:RESet
driver.sense.power.sweep.time.yscale.auto.reset()
```

Resets the Y scale to suitable values after the use of auto scaling in the expanding mode. For this mode, the scale might get expanded because of temporarily high power values. The reset function allows resetting the diagram to match smaller power values again.

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

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:AUTO:RESet
driver.sense.power.sweep.time.yscale.auto.reset_with_opc()
```

Resets the Y scale to suitable values after the use of auto scaling in the expanding mode. For this mode, the scale might get expanded because of temporarily high power values. The reset function allows resetting the diagram to match smaller power values again.

Same as reset, but waits for the operation to complete before continuing further. Use the RsSmab.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**(*auto: MeasRespYsCaleMode*) → None

```
# SCPI: SENSe:[POWer]:SWEep:TIME:YSCale:AUTO
driver.sense.power.sweep.time.yscale.auto.set_value(auto = enums.
↪MeasRespYsCaleMode.CEXPanding)
```

Activates autoscaling of the Y axis in the diagram.

**param auto**
OFF| CEXPanding| FEXPanding| CFLoating| FFLoating OFF Auto scaling is deactivated. When switching from activated to deactivated Auto scaling, the scaling is maintained. When switching from deactivated to activated Auto scaling, the scaling is reset to min = max = 0. CEXPanding | FEXPanding Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is expanded if the minimum or maximum values of the trace move outside the current scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine. CFLoating | FFLoating Auto scale is activated. The scaling of the Y-axis is selected in such a way, that the trace is always visible. To this end, the range is either expanded if the minimum or maximum values of the trace move outside the current scale or scaled down if the trace fits into a reduced scale. The step width is 5 dB for selection course and variable in the range of 0.2 db to 5 dB for selection fine.

#### 6.16.1.14 TypePy

##### SCPI Command :

```
SENSe<CH>:[POWer]:TYPE
```

**class TypePyCls**

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

**get**(*channel=Channel.Default*) → str

```
# SCPI: SENSe<CH>:[POWer]:TYPE
value: str = driver.sense.power.typePy.get(channel = repcap.Channel.Default)
```

Queries the sensor type. The type is automatically detected.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
type_py: string

#### 6.16.1.15 Zero

**SCPI Command :**

```
SENSe<CH>:[POWer]:ZERO
```

**class ZeroCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:ZERO
driver.sense.power.zero.set(channel = repcap.Channel.Default)
```

Performs zeroing of the sensor. Zeroing is required after warm-up, i.e. after connecting the sensor. Note: Switch off or disconnect the RF power source from the sensor before zeroing.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- The temperature has varied more than about 5 Deg.
- The sensor has been replaced.
- You want to measure very low power.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

### 6.16.2 Unit

**class UnitCls**

Unit 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.sense.unit.clone()
```

### Subgroups

#### 6.16.2.1 Power

### SCPI Command :

```
SENSe<CH>:UNIT:[POWer]
```

**class PowerCls**

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

**get**(*channel=Channel.Default*) → UnitPowSens

```
# SCPI: SENSe<CH>:UNIT:[POWer]
value: enums.UnitPowSens = driver.sense.unit.power.get(channel = repcap.Channel.
↪Default)
```

Selects the unit (Watt, dBm or dBuV) of measurement result display, queried with method **RsSmab.Read.Power.get_**.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
power: DBM| DBUV| WATT

**set**(*power: UnitPowSens*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:UNIT:[POWer]
driver.sense.unit.power.set(power = enums.UnitPowSens.DBM, channel = repcap.
↪Channel.Default)
```

Selects the unit (Watt, dBm or dBuV) of measurement result display, queried with method **RsSmab.Read.Power.get_**.

**param power**
DBM| DBUV| WATT

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

## 6.17 Slist

### SCPI Commands :

```
SLISt:CLEar:[ALL]
SLISt:[LIST]
```

**class SlistCls**

Slist commands group definition. 9 total commands, 4 Subgroups, 2 group commands

**clear_all**() → None

```
# SCPI: SLISt:CLEar:[ALL]
driver.slist.clear_all()
```

Removes all R&S NRP power sensors from the list.

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

```
# SCPI: SLISt:CLEar:[ALL]
driver.slist.clear_all_with_opc()
```

Removes all R&S NRP power sensors from the list.

Same as clear_all, but waits for the operation to complete before continuing further. Use the RsSmab.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**() → List[str]

```
# SCPI: SLISt:[LIST]
value: List[str] = driver.slist.get_list_py()
```

Returns a list of all detected sensors in a comma-separated string.

**return**
sensor_list: String of comma-separated entries Each entry contains information on the sensor type, serial number and interface. The order of the entries does not correspond to the order the sensors are displayed in the 'NRP Sensor Mapping' dialog.

### Cloning the Group

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

### Subgroups

## 6.17.1 Clear

**class ClearCls**

Clear 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.slist.clear.clone()
```

### Subgroups

#### 6.17.1.1 Lan

### SCPI Command :

```
SLISt:CLEar:LAN
```

**class LanCls**

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

**set**() → None

```
# SCPI: SLISt:CLEar:LAN
driver.slist.clear.lan.set()
```

Removes all R&S NRP power sensors connected in the LAN from the list.

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

```
# SCPI: SLISt:CLEar:LAN
driver.slist.clear.lan.set_with_opc()
```

Removes all R&S NRP power sensors connected in the LAN from the list.

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

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

#### 6.17.1.2 Usb

##### SCPI Command :

```
SLISt:CLEar:USB
```

**class UsbCls**

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

**set**() → None

```
# SCPI: SLISt:CLEar:USB
driver.slist.clear.usb.set()
```

Removes all R&S NRP power sensors connected over USB from the list.

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

```
# SCPI: SLISt:CLEar:USB
driver.slist.clear.usb.set_with_opc()
```

Removes all R&S NRP power sensors connected over USB from the list.

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

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

### 6.17.2 Element<Channel>

##### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.slist.element.repcap_channel_get()
driver.slist.element.repcap_channel_set(repcap.Channel.Nr1)
```

**class ElementCls**

Element commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

##### Cloning the Group

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

#### Subgroups

##### 6.17.2.1 Mapping

#### SCPI Command :

```
SLISt:ELEMent<CH>:MAPPing
```

**class MappingCls**

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

**get**(*channel=Channel.Default*) → ErFpowSensMapping

```
# SCPI: SLISt:ELEMent<CH>:MAPPing
value: enums.ErFpowSensMapping = driver.slist.element.mapping.get(channel =␣
↪repcap.Channel.Default)
```

Assigns an entry from the method RsSmab.Slist.listPy to one of the four sensor channels.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Element')

**return**
mapping: SENS1| SENSor1| SENS2| SENSor2| SENS3| SENSor3| SENS4| SENSor4| UNMapped Sensor channel.

**set**(*mapping: ErFpowSensMapping*, *channel=Channel.Default*) → None

```
# SCPI: SLISt:ELEMent<CH>:MAPPing
driver.slist.element.mapping.set(mapping = enums.ErFpowSensMapping.SENS1,␣
↪channel = repcap.Channel.Default)
```

Assigns an entry from the method RsSmab.Slist.listPy to one of the four sensor channels.

**param mapping**
SENS1| SENSor1| SENS2| SENSor2| SENS3| SENSor3| SENS4| SENSor4| UNMapped Sensor channel.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Element')

### 6.17.3 Scan

#### SCPI Commands :

```
SLISt:SCAN:LSENsor
SLISt:SCAN:[STATe]
```

**class ScanCls**

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

**get_state**() → bool

```
# SCPI: SLISt:SCAN:[STATe]
value: bool = driver.slist.scan.get_state()
```

Starts the search for R&S NRP power sensors, connected in the LAN or via the USBTMC protocol.

**return**
state: 1| ON| 0| OFF

**set_lsensor**(*ip: str*) → None

```
# SCPI: SLISt:SCAN:LSENsor
driver.slist.scan.set_lsensor(ip = 'abc')
```

Scans for R&S NRP power sensors connected in the LAN.

**param ip**
string

**set_state**(*state: bool*) → None

```
# SCPI: SLISt:SCAN:[STATe]
driver.slist.scan.set_state(state = False)
```

Starts the search for R&S NRP power sensors, connected in the LAN or via the USBTMC protocol.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.17.3.1 Usensor

### SCPI Command :

```
SLISt:SCAN:USENsor
```

**class UsensorCls**

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

**set**(*device_id: str*, *serial: int*) → None

```
# SCPI: SLISt:SCAN:USENsor
driver.slist.scan.usensor.set(device_id = 'abc', serial = 1)
```

Scans for R&S NRP power sensors connected over a USB interface.

**param device_id**
String or Integer Range: 0 to 999999

**param serial**
integer Range: 0 to 999999

### 6.17.4 Sensor

**class SensorCls**

Sensor 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.slist.sensor.clone()
```

#### Subgroups

##### 6.17.4.1 Map

#### SCPI Command :

```
SLISt:SENSor:MAP
```

**class MapCls**

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

**set**(*sensor_id: str*, *mapping: ErFpowSensMapping*) → None

```
# SCPI: SLISt:SENSor:MAP
driver.slist.sensor.map.set(sensor_id = 'abc', mapping = enums.
↪ErFpowSensMapping.SENS1)
```

Assigns a sensor directly to one of the sensor channels, using the sensor name and serial number. To find out the the sensor name and ID, you can get it from the label of the R&S NRP, or using the command method RsSmab.Slist.Scan.state. This command detects all R&S NRP power sensors connected in the LAN or via 'USBTMC protocol.

**param sensor_id**
string

**param mapping**
enum

## 6.18 Source

#### SCPI Command :

```
SOURce<HW>:PRESet
```

**class SourceCls**

Source commands group definition. 506 total commands, 29 Subgroups, 1 group commands

**preset**() → None

```
# SCPI: SOURce<HW>:PRESet
driver.source.preset()
```

Presets all parameters which are related to the selected signal path.

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

```
# SCPI: SOURce<HW>:PRESet
driver.source.preset_with_opc()
```

Presets all parameters which are related to the selected signal path.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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.source.clone()
```

#### Subgroups

### 6.18.1 Adf

#### SCPI Commands :

```
[SOURce<HW>]:ADF:PRESet
[SOURce<HW>]:ADF:STATe
```

**class AdfCls**

Adf commands group definition. 16 total commands, 2 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:ADF:STATe
value: bool = driver.source.adf.get_state()
```

Activates/deactivates the VOR modulation.

**return**
state: 1| ON| 0| OFF

**preset**() → None

```
# SCPI: [SOURce<HW>]:ADF:PRESet
driver.source.adf.preset()
```

Sets the parameters of the digital standard to their default values (*RST values specified for the commands) . Not affected is the state set with the command SOURce<hw>:VOR:STATe.

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

```
# SCPI: [SOURce<HW>]:ADF:PRESet
driver.source.adf.preset_with_opc()
```

Sets the parameters of the digital standard to their default values (*RST values specified for the commands) . Not affected is the state set with the command SOURce<hw>:VOR:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:ADF:STATe
driver.source.adf.set_state(state = False)
```

Activates/deactivates the VOR modulation.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.1.1 Comid

### SCPI Commands :

```
[SOURce<HW>]:ADF:COMid:CODE
[SOURce<HW>]:ADF:COMid:DASH
[SOURce<HW>]:ADF:COMid:DEPTh
[SOURce<HW>]:ADF:COMid:DOT
[SOURce<HW>]:ADF:COMid:FREQuency
[SOURce<HW>]:ADF:COMid:LETTer
[SOURce<HW>]:ADF:COMid:PERiod
[SOURce<HW>]:ADF:COMid:SYMBol
[SOURce<HW>]:ADF:COMid:TSCHema
[SOURce<HW>]:ADF:COMid:[STATe]
```

**class ComidCls**

Comid commands group definition. 10 total commands, 0 Subgroups, 10 group commands

**get_code**() → str

```
# SCPI: [SOURce<HW>]:ADF:COMid:CODE
value: str = driver.source.adf.comid.get_code()
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**return**
code: string

**get_dash**() → float

```
# SCPI: [SOURce<HW>]:ADF:COMid:DASH
value: float = driver.source.adf.comid.get_dash()
```

Sets the length of a Morse code dash.

**return**
dash: float Range: 50E-3 to 1

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:ADF:COMid:DEPTh
value: float = driver.source.adf.comid.get_depth()
```

Sets the AM modulation depth of the COM/ID signal.

**return**
depth: float Range: 0 to 100

**get_dot**() → float

```
# SCPI: [SOURce<HW>]:ADF:COMid:DOT
value: float = driver.source.adf.comid.get_dot()
```

Sets the length of a Morse code dot.

**return**
dot: float Range: 50E-3 to 1

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ADF:COMid:FREQuency
value: float = driver.source.adf.comid.get_frequency()
```

Sets the frequency of the COM/ID signal.

**return**
frequency: float Range: 0.1 to 20E3

**get_letter**() → float

```
# SCPI: [SOURce<HW>]:ADF:COMid:LETTer
value: float = driver.source.adf.comid.get_letter()
```

Sets the length of a Morse code letter space.

**return**
letter: float Range: 50E-3 to 1

**get_period**() → float

```
# SCPI: [SOURce<HW>]:ADF:COMid:PERiod
value: float = driver.source.adf.comid.get_period()
```

Sets the period of the COM/ID signal.

**return**
period: float Range: 0 to 120

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:ADF:COMid:[STATe]
value: bool = driver.source.adf.comid.get_state()
```

Enables/disables the COM/ID signal.

**return**
state: 1| ON| 0| OFF

**get_symbol**() → float

```
# SCPI: [SOURce<HW>]:ADF:COMid:SYMBol
value: float = driver.source.adf.comid.get_symbol()
```

Sets the length of the Morse code symbol space.

**return**
symbol: float Range: 50E-3 to 1

**get_tschema**() → AvionicComIdTimeSchem

```
# SCPI: [SOURce<HW>]:ADF:COMid:TSCHema
value: enums.AvionicComIdTimeSchem = driver.source.adf.comid.get_tschema()
```

Sets the time schema of the Morse code for the COM/ID signal.

**return**
tschema: STD| USER

**set_code**(*code: str*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:CODE
driver.source.adf.comid.set_code(code = 'abc')
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**param code**
string

**set_dash**(*dash: float*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:DASH
driver.source.adf.comid.set_dash(dash = 1.0)
```

Sets the length of a Morse code dash.

**param dash**
float Range: 50E-3 to 1

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:DEPTh
driver.source.adf.comid.set_depth(depth = 1.0)
```

Sets the AM modulation depth of the COM/ID signal.

**param depth**
float Range: 0 to 100

**set_dot**(*dot: float*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:DOT
driver.source.adf.comid.set_dot(dot = 1.0)
```

Sets the length of a Morse code dot.

**param dot**
float Range: 50E-3 to 1

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:FREQuency
driver.source.adf.comid.set_frequency(frequency = 1.0)
```

Sets the frequency of the COM/ID signal.

**param frequency**
float Range: 0.1 to 20E3

**set_letter**(*letter: float*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:LETTer
driver.source.adf.comid.set_letter(letter = 1.0)
```

Sets the length of a Morse code letter space.

**param letter**
float Range: 50E-3 to 1

**set_period**(*period: float*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:PERiod
driver.source.adf.comid.set_period(period = 1.0)
```

Sets the period of the COM/ID signal.

**param period**
float Range: 0 to 120

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:[STATe]
driver.source.adf.comid.set_state(state = False)
```

Enables/disables the COM/ID signal.

**param state**
1| ON| 0| OFF

**set_symbol**(*symbol: float*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:SYMBol
driver.source.adf.comid.set_symbol(symbol = 1.0)
```

Sets the length of the Morse code symbol space.

**param symbol**
float Range: 50E-3 to 1

**set_tschema**(*tschema: AvionicComIdTimeSchem*) → None

```
# SCPI: [SOURce<HW>]:ADF:COMid:TSCHema
driver.source.adf.comid.set_tschema(tschema = enums.AvionicComIdTimeSchem.STD)
```

Sets the time schema of the Morse code for the COM/ID signal.

**param tschema**
STD| USER

#### 6.18.1.2 Setting

**SCPI Commands :**

```
[SOURce<HW>]:ADF:SETTing:CATalog
[SOURce<HW>]:ADF:SETTing:DELete
[SOURce<HW>]:ADF:SETTing:LOAD
[SOURce<HW>]:ADF:SETTing:STORe
```

**class SettingCls**

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

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:ADF:SETTing:DELete
driver.source.adf.setting.delete(filename = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path; file extension can be omitted

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:ADF:SETTing:CATalog
value: List[str] = driver.source.adf.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**return**
avionic_adf_cat_names: filename1,filename2,... Returns a string of filenames separated by commas.

**load**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:ADF:SETTing:LOAD
driver.source.adf.setting.load(filename = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path; file extension can be omitted

**set_store**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:ADF:SETTing:STORe
driver.source.adf.setting.set_store(filename = 'abc')
```

Saves the current settings into the selected file; the file extension (*.adf/*.ils/*.vor) is assigned automatically. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path

### 6.18.2 Am<GeneratorIx>

#### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.am.repcap_generatorIx_get()
driver.source.am.repcap_generatorIx_set(repcap.GeneratorIx.Nr1)
```

#### SCPI Commands :

```
[SOURce<HW>]:AM:MODE
[SOURce<HW>]:AM:RATio
[SOURce<HW>]:AM:TYPE
```

**class AmCls**

Am commands group definition. 11 total commands, 4 Subgroups, 3 group commands Repeated Capability: GeneratorIx, default value after init: GeneratorIx.Nr1

**get_mode**() → AmMode

```
# SCPI: [SOURce<HW>]:AM:MODE
value: enums.AmMode = driver.source.am.get_mode()
```

Selects the mode of the amplitude modulation. [:SOURce<hw>]:AM:MODE > SCAN sets [:SOURce<hw>]:AM:TYPE > EXPonential. For active external exponential AM, automatically sets [:SOURce<hw>]:INPut:MODext:COUPling<ch> > DC.

**return**
am_mode: SCAN| NORMal

**get_ratio**() → float

```
# SCPI: [SOURce<HW>]:AM:RATio
value: float = driver.source.am.get_ratio()
```

Sets the deviation ratio (path#2 to path#1) in percent.

**return**
ratio: float Range: 0 to 100

**get_type_py**() → AmType

```
# SCPI: [SOURce<HW>]:AM:TYPE
value: enums.AmType = driver.source.am.get_type_py()
```

Selects the type of amplitude modulation. For [:SOURce<hw>]:AM:MODE SCAN, only EXPonential is available. For active external exponential AM, automatically sets [:SOURce<hw>]:INPut:MODext:COUPling<ch>DC.

**return**
am_type: LINear| EXPonential

**set_mode**(*am_mode: AmMode*) → None

```
# SCPI: [SOURce<HW>]:AM:MODE
driver.source.am.set_mode(am_mode = enums.AmMode.NORMal)
```

Selects the mode of the amplitude modulation. [:SOURce<hw>]:AM:MODE > SCAN sets [:SOURce<hw>]:AM:TYPE > EXPonential. For active external exponential AM, automatically sets [:SOURce<hw>]:INPut:MODext:COUPling<ch> > DC.

**param am_mode**
SCAN| NORMal

**set_ratio**(*ratio: float*) → None

```
# SCPI: [SOURce<HW>]:AM:RATio
driver.source.am.set_ratio(ratio = 1.0)
```

Sets the deviation ratio (path#2 to path#1) in percent.

**param ratio**
float Range: 0 to 100

**set_type_py**(*am_type: AmType*) → None

```
# SCPI: [SOURce<HW>]:AM:TYPE
driver.source.am.set_type_py(am_type = enums.AmType.EXPonential)
```

Selects the type of amplitude modulation. For [:SOURce<hw>]:AM:MODE SCAN, only EXPonential is available. For active external exponential AM, automatically sets [:SOURce<hw>]:INPut:MODext:COUPling<ch>DC.

**param am_type**
LINear| EXPonential

### Cloning the Group

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

### Subgroups

#### 6.18.2.1 Depth

### SCPI Commands :

```
[SOURce<HW>]:AM:DEPTh:SUM
[SOURce<HW>]:AM<CH>:[DEPTh]
```

**class DepthCls**

Depth commands group definition. 4 total commands, 2 Subgroups, 2 group commands

**get**(*generatorIx=GeneratorIx.Default*) → float

```
# SCPI: [SOURce<HW>]:AM<CH>:[DEPTh]
value: float = driver.source.am.depth.get(generatorIx = repcap.GeneratorIx.
↪Default)
```

Sets the depth of the amplitude modulation in percent.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

**return**
depth: float Range: 0 to 100

**get_sum**() → float

```
# SCPI: [SOURce<HW>]:AM:DEPTh:SUM
value: float = driver.source.am.depth.get_sum()
```

Sets the total depth of the LF signal when using combined signal sources in amplitude modulation.

**return**
am_depth_sum: float Range: 0 to 100

**set**(*depth: float*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:AM<CH>:[DEPTh]
driver.source.am.depth.set(depth = 1.0, generatorIx = repcap.GeneratorIx.
↪Default)
```

Sets the depth of the amplitude modulation in percent.

**param depth**
float Range: 0 to 100

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

**set_sum**(*am_depth_sum: float*) → None

```
# SCPI: [SOURce<HW>]:AM:DEPTh:SUM
driver.source.am.depth.set_sum(am_depth_sum = 1.0)
```

Sets the total depth of the LF signal when using combined signal sources in amplitude modulation.

**param am_depth_sum**
float Range: 0 to 100

### Cloning the Group

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

### Subgroups

#### 6.18.2.1.1 Exponential

### SCPI Command :

```
[SOURce<HW>]:AM<CH>:DEPTh:EXPonential
```

**class ExponentialCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → float

```
# SCPI: [SOURce<HW>]:AM<CH>:DEPTh:EXPonential
value: float = driver.source.am.depth.exponential.get(generatorIx = repcap.
↪GeneratorIx.Default)
```

Sets the depth of the exponential amplitude modulation in dB/volt.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

**return**
depth_exp: float Range: 0 to 100

**set**(*depth_exp: float*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:AM<CH>:DEPTh:EXPonential
driver.source.am.depth.exponential.set(depth_exp = 1.0, generatorIx = repcap.
↪GeneratorIx.Default)
```

Sets the depth of the exponential amplitude modulation in dB/volt.

**param depth_exp**
float Range: 0 to 100

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

#### 6.18.2.1.2 Linear

**SCPI Command :**

```
[SOURce<HW>]:AM<CH>:DEPTh:LINear
```

**class LinearCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → float

```
# SCPI: [SOURce<HW>]:AM<CH>:DEPTh:LINear
value: float = driver.source.am.depth.linear.get(generatorIx = repcap.
↪GeneratorIx.Default)
```

Sets the depth of the linear amplitude modulation in percent / volt.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

**return**
depth_lin: float Range: 0 to 100

**set**(*depth_lin: float*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:AM<CH>:DEPTh:LINear
driver.source.am.depth.linear.set(depth_lin = 1.0, generatorIx = repcap.
↪GeneratorIx.Default)
```

Sets the depth of the linear amplitude modulation in percent / volt.

**param depth_lin**
float Range: 0 to 100

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

### 6.18.2.2 Deviation

**SCPI Command :**

```
[SOURce<HW>]:AM:DEViation:MODE
```

**class DeviationCls**

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

**get_mode**() → ModulationDevMode

```
# SCPI: [SOURce<HW>]:AM:DEViation:MODE
value: enums.ModulationDevMode = driver.source.am.deviation.get_mode()
```

Selects the coupling mode. The coupling mode parameter also determines the mode for fixing the total depth.

**return**
am_dev_mode: UNCoupled| TOTal| RATio UNCoupled Does not couple the LF signals. The deviation depth values of both paths are independent. TOTal Couples the deviation depth of both paths. RATio Couples the deviation depth ratio of both paths

**set_mode**(*am_dev_mode: ModulationDevMode*) → None

```
# SCPI: [SOURce<HW>]:AM:DEViation:MODE
driver.source.am.deviation.set_mode(am_dev_mode = enums.ModulationDevMode.RATio)
```

Selects the coupling mode. The coupling mode parameter also determines the mode for fixing the total depth.

**param am_dev_mode**
UNCoupled| TOTal| RATio UNCoupled Does not couple the LF signals. The deviation depth values of both paths are independent. TOTal Couples the deviation depth of both paths. RATio Couples the deviation depth ratio of both paths

#### 6.18.2.3 Sensitivity

**class SensitivityCls**

Sensitivity 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.source.am.sensitivity.clone()
```

##### Subgroups

###### 6.18.2.3.1 Exponential

##### SCPI Command :

```
[SOURce<HW>]:AM<CH>:SENSitivity:EXPonential
```

**class ExponentialCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → float

```
# SCPI: [SOURce<HW>]:AM<CH>:SENSitivity:EXPonential
value: float = driver.source.am.sensitivity.exponential.get(generatorIx =␣
↪repcap.GeneratorIx.Default)
```

For [:SOURce<hw>]:AM:TYPEEXP, sets the sensitivity of the external signal source for amplitude modulation.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

**return**
sensitivity: float Range: 0 to 100

#### 6.18.2.3.2 Linear

**SCPI Command :**

```
[SOURce<HW>]:AM<CH>:SENSitivity:[LINear]
```

**class LinearCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → float

```
# SCPI: [SOURce<HW>]:AM<CH>:SENSitivity:[LINear]
value: float = driver.source.am.sensitivity.linear.get(generatorIx = repcap.
↪GeneratorIx.Default)
```

For [:SOURce<hw>]:AM:TYPE LIN, sets the sensitivity of the external signal source for amplitude modulation.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

**return**
sensitivity: float Range: 0 to 100

### 6.18.2.4 State

**SCPI Command :**

```
[SOURce<HW>]:AM<CH>:STATe
```

**class StateCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → bool

```
# SCPI: [SOURce<HW>]:AM<CH>:STATe
value: bool = driver.source.am.state.get(generatorIx = repcap.GeneratorIx.
↪Default)
```

Activates amplitude modulation.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:AM<CH>:STATe
driver.source.am.state.set(state = False, generatorIx = repcap.GeneratorIx.
↪Default)
```

Activates amplitude modulation.

**param state**
1| ON| 0| OFF

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Am')

### 6.18.3 Bb

**class BbCls**

Bb commands group definition. 17 total commands, 4 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.18.3.1 Dme

#### SCPI Commands :

```
[SOURce<HW>]:BB:DME:PRESet
[SOURce<HW>]:BB:DME:STATe
```

**class DmeCls**

Dme commands group definition. 6 total commands, 2 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:BB:DME:STATe
value: bool = driver.source.bb.dme.get_state()
```

No command help available

**return**
state: No help available

**preset**() → None

```
# SCPI: [SOURce<HW>]:BB:DME:PRESet
driver.source.bb.dme.preset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:DME:PRESet
driver.source.bb.dme.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DME:STATe
driver.source.bb.dme.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.3.1.1 Gaussian

### SCPI Command :

```
[SOURce<HW>]:[BB]:DME:GAUSsian
```

**class GaussianCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:[BB]:DME:GAUSsian
driver.source.bb.dme.gaussian.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:[BB]:DME:GAUSsian
driver.source.bb.dme.gaussian.set_with_opc()
```

No command help available

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

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

#### 6.18.3.1.2 Setting

##### SCPI Commands :

```
[SOURce<HW>]:BB:DME:SETTing:DELete
[SOURce<HW>]:BB:DME:SETTing:LOAD
[SOURce<HW>]:BB:DME:SETTing:STORe
```

**class SettingCls**

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

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DME:SETTing:DELete
driver.source.bb.dme.setting.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**load**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DME:SETTing:LOAD
driver.source.bb.dme.setting.load(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_store**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DME:SETTing:STORe
driver.source.bb.dme.setting.set_store(filename = 'abc')
```

No command help available

**param filename**
No help available

#### 6.18.3.2 Ils

**SCPI Commands :**

```
[SOURce<HW>]:[BB]:ILS:PRESet
[SOURce<HW>]:[BB]:ILS:STATe
```

**class IlsCls**

Ils commands group definition. 5 total commands, 1 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:[BB]:ILS:STATe
value: bool = driver.source.bb.ils.get_state()
```

No command help available

**return**
state: No help available

**preset**() → None

```
# SCPI: [SOURce<HW>]:[BB]:ILS:PRESet
driver.source.bb.ils.preset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:[BB]:ILS:PRESet
driver.source.bb.ils.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:[BB]:ILS:STATe
driver.source.bb.ils.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.3.2.1 Setting

### SCPI Commands :

```
[SOURce<HW>]:[BB]:ILS:SETTing:DELete
[SOURce<HW>]:[BB]:ILS:SETTing:LOAD
[SOURce<HW>]:[BB]:ILS:SETTing:STORe
```

**class SettingCls**

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

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:[BB]:ILS:SETTing:DELete
driver.source.bb.ils.setting.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**load**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:[BB]:ILS:SETTing:LOAD
driver.source.bb.ils.setting.load(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_store**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:[BB]:ILS:SETTing:STORe
driver.source.bb.ils.setting.set_store(filename = 'abc')
```

No command help available

**param filename**
No help available

#### 6.18.3.3 Path

**SCPI Command :**

```
[SOURce]:BB:PATH:COUNt
```

**class PathCls**

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

**get_count**() → int

```
# SCPI: [SOURce]:BB:PATH:COUNt
value: int = driver.source.bb.path.get_count()
```

No command help available

**return**
count: No help available

#### 6.18.3.4 Vor

**SCPI Commands :**

```
[SOURce<HW>]:BB:VOR:PRESet
[SOURce<HW>]:BB:VOR:STATe
```

**class VorCls**

Vor commands group definition. 5 total commands, 1 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:BB:VOR:STATe
value: bool = driver.source.bb.vor.get_state()
```

No command help available

**return**
state: No help available

**preset**() → None

```
# SCPI: [SOURce<HW>]:BB:VOR:PRESet
driver.source.bb.vor.preset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:VOR:PRESet
driver.source.bb.vor.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:VOR:STATe
driver.source.bb.vor.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.3.4.1 Setting

### SCPI Commands :

```
[SOURce<HW>]:BB:VOR:SETTing:DELete
[SOURce<HW>]:BB:VOR:SETTing:LOAD
[SOURce<HW>]:BB:VOR:SETTing:STORe
```

**class SettingCls**

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

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:VOR:SETTing:DELete
driver.source.bb.vor.setting.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**load**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:VOR:SETTing:LOAD
driver.source.bb.vor.setting.load(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_store**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:VOR:SETTing:STORe
driver.source.bb.vor.setting.set_store(filename = 'abc')
```

No command help available

**param filename**
No help available

### 6.18.4 Chirp

#### SCPI Commands :

```
[SOURce<HW>]:CHIRp:BANDwidth
[SOURce<HW>]:CHIRp:DIRection
[SOURce<HW>]:CHIRp:STATe
```

**class ChirpCls**

Chirp commands group definition. 10 total commands, 4 Subgroups, 3 group commands

**get_bandwidth**() → float

```
# SCPI: [SOURce<HW>]:CHIRp:BANDwidth
value: float = driver.source.chirp.get_bandwidth()
```

Sets the modulation bandwidth of the chirp modulated signal.

**return**
bandwidth: float Range: 0 to Depends on hardware variant

**get_direction**() → UpDownDirection

```
# SCPI: [SOURce<HW>]:CHIRp:DIRection
value: enums.UpDownDirection = driver.source.chirp.get_direction()
```

Selects the direction of the chirp modulation.

**return**
direction: DOWN| UP

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:CHIRp:STATe
value: bool = driver.source.chirp.get_state()
```

Activates the generation of a chirp modulation signal.

**return**
state: 1| ON| 0| OFF

**set_bandwidth**(*bandwidth: float*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:BANDwidth
driver.source.chirp.set_bandwidth(bandwidth = 1.0)
```

Sets the modulation bandwidth of the chirp modulated signal.

**param bandwidth**
float Range: 0 to Depends on hardware variant

**set_direction**(*direction: UpDownDirection*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:DIRection
driver.source.chirp.set_direction(direction = enums.UpDownDirection.DOWN)
```

Selects the direction of the chirp modulation.

**param direction**
DOWN| UP

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:STATe
driver.source.chirp.set_state(state = False)
```

Activates the generation of a chirp modulation signal.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.4.1 Compression

### SCPI Command :

```
[SOURce<HW>]:CHIRp:COMPression:RATio
```

**class CompressionCls**

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

**get_ratio**() → float

```
# SCPI: [SOURce<HW>]:CHIRp:COMPression:RATio
value: float = driver.source.chirp.compression.get_ratio()
```

Queries the pulse compression ratio (= product of pulse width (s) and bandwidth (Hz) ).

**return**
ratio: float Range: 0 to 80E6

#### 6.18.4.2 Pulse

**SCPI Commands :**

```
[SOURce<HW>]:CHIRp:PULSe:NUMBer
[SOURce<HW>]:CHIRp:PULSe:PERiod
[SOURce<HW>]:CHIRp:PULSe:WIDTh
```

**class PulseCls**

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

**get_number**() → int

```
# SCPI: [SOURce<HW>]:CHIRp:PULSe:NUMBer
value: int = driver.source.chirp.pulse.get_number()
```

No command help available

**return**
number_of_pulses: No help available

**get_period**() → float

```
# SCPI: [SOURce<HW>]:CHIRp:PULSe:PERiod
value: float = driver.source.chirp.pulse.get_period()
```

Sets the period of the generated modulation chirp. The period determines the repetition frequency of the internal signal.

**return**
period: float Range: 5E-6 (2E-7 with K23) to 100

**get_width**() → float

```
# SCPI: [SOURce<HW>]:CHIRp:PULSe:WIDTh
value: float = driver.source.chirp.pulse.get_width()
```

Sets the width of the generated pulse. The pulse width must be at least 1us less than the set pulse period.

**return**
width: float Range: 2E-6 (1E-7 with K23) to 100

**set_number**(*number_of_pulses: int*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:PULSe:NUMBer
driver.source.chirp.pulse.set_number(number_of_pulses = 1)
```

No command help available

**param number_of_pulses**
No help available

**set_period**(*period: float*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:PULSe:PERiod
driver.source.chirp.pulse.set_period(period = 1.0)
```

Sets the period of the generated modulation chirp. The period determines the repetition frequency of the internal signal.

**param period**
float Range: 5E-6 (2E-7 with K23) to 100

**set_width**(*width: float*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:PULSe:WIDTh
driver.source.chirp.pulse.set_width(width = 1.0)
```

Sets the width of the generated pulse. The pulse width must be at least 1us less than the set pulse period.

**param width**
float Range: 2E-6 (1E-7 with K23) to 100

#### 6.18.4.3 Test

**class TestCls**

Test 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.source.chirp.test.clone()
```

##### Subgroups

###### 6.18.4.3.1 Measurement

##### SCPI Command :

```
[SOURce<HW>]:CHIRp:TEST:MEASurement:DELay
```

**class MeasurementCls**

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

**get_delay**() → bool

```
# SCPI: [SOURce<HW>]:CHIRp:TEST:MEASurement:DELay
value: bool = driver.source.chirp.test.measurement.get_delay()
```

No command help available

**return**
state: No help available

**set_delay**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:TEST:MEASurement:DELay
driver.source.chirp.test.measurement.set_delay(state = False)
```

No command help available

**param state**
No help available

#### 6.18.4.4 Trigger

##### SCPI Command :

```
[SOURce<HW>]:CHIRp:TRIGger:MODE
```

**class TriggerCls**

Trigger commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_mode**() → PulsTrigModeWithSingle

```
# SCPI: [SOURce<HW>]:CHIRp:TRIGger:MODE
value: enums.PulsTrigModeWithSingle = driver.source.chirp.trigger.get_mode()
```

Selects the trigger mode for the chirp modulation signal.

**return**
mode: AUTO| EXTernal| EGATe| SINGle| ESINgle

**set_mode**(*mode: PulsTrigModeWithSingle*) → None

```
# SCPI: [SOURce<HW>]:CHIRp:TRIGger:MODE
driver.source.chirp.trigger.set_mode(mode = enums.PulsTrigModeWithSingle.AUTO)
```

Selects the trigger mode for the chirp modulation signal.

**param mode**
AUTO| EXTernal| EGATe| SINGle| ESINgle

##### Cloning the Group

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

##### Subgroups

###### 6.18.4.4.1 Immediate

##### SCPI Command :

```
[SOURce<HW>]:CHIRp:TRIGger:IMMediate
```

**class ImmediateCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:CHIRp:TRIGger:IMMediate
driver.source.chirp.trigger.immediate.set()
```

Immediately starts the chirp signal generation.

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

```
# SCPI: [SOURce<HW>]:CHIRp:TRIGger:IMMediate
driver.source.chirp.trigger.immediate.set_with_opc()
```

Immediately starts the chirp signal generation.

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

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

### 6.18.5 Combined

**class CombinedCls**

Combined 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.source.combined.clone()
```

#### Subgroups

##### 6.18.5.1 Frequency

#### SCPI Commands :

```
[SOURce<HW>]:COMBined:FREQuency:STARt
[SOURce<HW>]:COMBined:FREQuency:STOP
```

**class FrequencyCls**

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

**get_start**() → float

```
# SCPI: [SOURce<HW>]:COMBined:FREQuency:STARt
value: float = driver.source.combined.frequency.get_start()
```

Sets the start frequency of the combined RF frequency / level sweep. See 'Correlating parameters in sweep mode'.

**return**
comb_freq_start: float Range: -59999E5 to 12E9

**get_stop**() → float

```
# SCPI: [SOURce<HW>]:COMBined:FREQuency:STOP
value: float = driver.source.combined.frequency.get_stop()
```

Sets the end frequency of the combined RF frequency / level sweep.

**return**
comb_freq_stop: float Range: -59999E5 to 12E9

**set_start**(*comb_freq_start: float*) → None

```
# SCPI: [SOURce<HW>]:COMBined:FREQuency:STARt
driver.source.combined.frequency.set_start(comb_freq_start = 1.0)
```

Sets the start frequency of the combined RF frequency / level sweep. See 'Correlating parameters in sweep mode'.

**param comb_freq_start**
float Range: -59999E5 to 12E9

**set_stop**(*comb_freq_stop: float*) → None

```
# SCPI: [SOURce<HW>]:COMBined:FREQuency:STOP
driver.source.combined.frequency.set_stop(comb_freq_stop = 1.0)
```

Sets the end frequency of the combined RF frequency / level sweep.

**param comb_freq_stop**
float Range: -59999E5 to 12E9

#### 6.18.5.2 Power

**SCPI Commands :**

```
[SOURce<HW>]:COMBined:POWer:STARt
[SOURce<HW>]:COMBined:POWer:STOP
```

**class PowerCls**

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

**get_start**() → float

```
# SCPI: [SOURce<HW>]:COMBined:POWer:STARt
value: float = driver.source.combined.power.get_start()
```

Sets the start level value of the combined RF frequency / level sweep. See 'Correlating parameters in sweep mode'.

**return**
comb_pow_start: float Range: -245 to 120

**get_stop**() → float

```
# SCPI: [SOURce<HW>]:COMBined:POWer:STOP
value: float = driver.source.combined.power.get_stop()
```

Sets the stop level value of the combined RF frequency / level sweep.

**return**
comb_pow_stop: float Range: -245 to 120

**set_start**(*comb_pow_start: float*) → None

```
# SCPI: [SOURce<HW>]:COMBined:POWer:STARt
driver.source.combined.power.set_start(comb_pow_start = 1.0)
```

Sets the start level value of the combined RF frequency / level sweep. See 'Correlating parameters in sweep mode'.

**param comb_pow_start**
float Range: -245 to 120

**set_stop**(*comb_pow_stop: float*) → None

```
# SCPI: [SOURce<HW>]:COMBined:POWer:STOP
driver.source.combined.power.set_stop(comb_pow_stop = 1.0)
```

Sets the stop level value of the combined RF frequency / level sweep.

**param comb_pow_stop**
float Range: -245 to 120

### 6.18.6 Correction

#### SCPI Commands :

```
[SOURce<HW>]:CORRection:VALue
[SOURce<HW>]:CORRection:[STATe]
```

**class CorrectionCls**

Correction commands group definition. 19 total commands, 3 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:CORRection:[STATe]
value: bool = driver.source.correction.get_state()
```

Activates user correction with the currently selected table.

**return**
state: 1| ON| 0| OFF

**get_value**() → float

```
# SCPI: [SOURce<HW>]:CORRection:VALue
value: float = driver.source.correction.get_value()
```

Queries the current value for user correction.

**return**
value: float Range: -100 to 100

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:CORRection:[STATe]
driver.source.correction.set_state(state = False)
```

Activates user correction with the currently selected table.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.6.1 Cset

### SCPI Commands :

```
[SOURce]:CORRection:CSET:CATalog
[SOURce]:CORRection:CSET:DELete
[SOURce<HW>]:CORRection:CSET:[SELect]
```

**class CsetCls**

Cset commands group definition. 8 total commands, 1 Subgroups, 3 group commands

**delete**(*filename: str*) → None

```
# SCPI: [SOURce]:CORRection:CSET:DELete
driver.source.correction.cset.delete(filename = 'abc')
```

Deletes the specified user correction list file.

**param filename**
string Filename or complete file path; file extension is optional.

**get_catalog**() → List[str]

```
# SCPI: [SOURce]:CORRection:CSET:CATalog
value: List[str] = driver.source.correction.cset.get_catalog()
```

Queries a list of available user correction tables.

**return**
catalog: string List of list filenames, separated by commas

**get_select**() → str

```
# SCPI: [SOURce<HW>]:CORRection:CSET:[SELect]
value: str = driver.source.correction.cset.get_select()
```

Selects or creates a file for the user correction data. If the file with the selected name does not exist, a new file is created.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:CORRection:CSET:[SELect]
driver.source.correction.cset.set_select(filename = 'abc')
```

Selects or creates a file for the user correction data. If the file with the selected name does not exist, a new file is created.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

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

### Subgroups

#### 6.18.6.1.1 Data

**class DataCls**

Data commands group definition. 5 total commands, 3 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.correction.cset.data.clone()
```

### Subgroups

#### 6.18.6.1.1.1 Frequency

### SCPI Commands :

```
[SOURce<HW>]:CORRection:CSET:DATA:FREQuency:POINts
[SOURce<HW>]:CORRection:CSET:DATA:FREQuency
```

**class FrequencyCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:CORRection:CSET:DATA:FREQuency:POINts
value: int = driver.source.correction.cset.data.frequency.get_points()
```

Queries the number of frequency/level values in the selected table.

**return**
points: integer Range: 0 to 10000

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:CORRection:CSET:DATA:FREQuency
value: List[float] = driver.source.correction.cset.data.frequency.get_value()
```

Enters the frequency value in the table selected with [:SOURce<hw>]:CORRection:CSET[:SELect].

**return**
frequency: Frequency#1[, Frequency#2, ...] String of values with default unit Hz.

**set_value**(*frequency: List[float]*) → None

```
# SCPI: [SOURce<HW>]:CORRection:CSET:DATA:FREQuency
driver.source.correction.cset.data.frequency.set_value(frequency = [1.1, 2.2, 3.
↪3])
```

Enters the frequency value in the table selected with [:SOURce<hw>]:CORRection:CSET[:SELect].

**param frequency**
Frequency#1[, Frequency#2, ...] String of values with default unit Hz.

#### 6.18.6.1.1.2 Power

**SCPI Commands :**

```
[SOURce<HW>]:CORRection:CSET:DATA:POWer:POINts
[SOURce<HW>]:CORRection:CSET:DATA:POWer
```

**class PowerCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:CORRection:CSET:DATA:POWer:POINts
value: int = driver.source.correction.cset.data.power.get_points()
```

Queries the number of frequency/level values in the selected table.

**return**
points: integer Range: 0 to 10000

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:CORRection:CSET:DATA:POWer
value: List[float] = driver.source.correction.cset.data.power.get_value()
```

Enters the level values to the table selected with [:SOURce<hw>]:CORRection:CSET[:SELect].

**return**
power: Power#1[, Power#2, ...] String of values with default unit dB. *RST: 0

**set_value**(*power: List[float]*) → None

```
# SCPI: [SOURce<HW>]:CORRection:CSET:DATA:POWer
driver.source.correction.cset.data.power.set_value(power = [1.1, 2.2, 3.3])
```

Enters the level values to the table selected with [:SOURce<hw>]:CORRection:CSET[:SELect].

**param power**
Power#1[, Power#2, ...] String of values with default unit dB. *RST: 0

#### 6.18.6.1.1.3 Sensor<Channel>

##### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.correction.cset.data.sensor.repcap_channel_get()
driver.source.correction.cset.data.sensor.repcap_channel_set(repcap.Channel.Nr1)
```

**class SensorCls**

Sensor commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.correction.cset.data.sensor.clone()
```

##### Subgroups

#### 6.18.6.1.1.4 Power

**class PowerCls**

Power 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.source.correction.cset.data.sensor.power.clone()
```

##### Subgroups

#### 6.18.6.1.1.5 Sonce

##### SCPI Command :

```
[SOURce<HW>]:CORRection:CSET:DATA:[SENSor<CH>]:[POWer]:SONCe
```

**class SonceCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:CORRection:CSET:DATA:[SENSor<CH>]:[POWer]:SONCe
driver.source.correction.cset.data.sensor.power.sonce.set(channel = repcap.
↪Channel.Default)
```

Fills the selected user correction table with the level values measured by the power sensor for the given frequencies. To select the used power sensor set the suffix in key word SENSe.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.18.6.2 Dexchange

**SCPI Commands :**

```
[SOURce<HW>]:CORRection:DEXChange:MODE
[SOURce<HW>]:CORRection:DEXChange:SELect
```

**class DexchangeCls**

Dexchange commands group definition. 8 total commands, 2 Subgroups, 2 group commands

**get_mode**() → DexchMode

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:MODE
value: enums.DexchMode = driver.source.correction.dexchange.get_mode()
```

Determines import or export of a user correction list. Specify the source or destination file with the command [:SOURce<hw>]:CORRection:DEXChange:SELect.

**return**
mode: IMPort| EXPort

**get_select**() → str

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:SELect
value: str = driver.source.correction.dexchange.get_select()
```

Selects the ASCII file for import or export, containing a user correction list.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_mode**(*mode: DexchMode*) → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:MODE
driver.source.correction.dexchange.set_mode(mode = enums.DexchMode.EXPort)
```

Determines import or export of a user correction list. Specify the source or destination file with the command [:SOURce<hw>]:CORRection:DEXChange:SELect.

**param mode**
IMPort| EXPort

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:SELect
driver.source.correction.dexchange.set_select(filename = 'abc')
```

Selects the ASCII file for import or export, containing a user correction list.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

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

### Subgroups

#### 6.18.6.2.1 Afile

### SCPI Commands :

```
[SOURce<HW>]:CORRection:DEXChange:AFILe:CATalog
[SOURce<HW>]:CORRection:DEXChange:AFILe:EXTension
[SOURce<HW>]:CORRection:DEXChange:AFILe:SELect
```

**class AfileCls**

Afile commands group definition. 5 total commands, 1 Subgroups, 3 group commands

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:CATalog
value: List[str] = driver.source.correction.dexchange.afile.get_catalog()
```

Queries the available ASCII files for export or import of user correction data in the current or specified directory.

**return**
catalog: string List of ASCII files *.txt or *.csv, separated by commas.

**get_extension**() → DexchExtension

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:EXTension
value: enums.DexchExtension = driver.source.correction.dexchange.afile.get_
↪extension()
```

Determines the extension of the ASCII files for file import or export, or to query existing files.

**return**
extension: TXT| CSV

**get_select**() → str

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SELect
value: str = driver.source.correction.dexchange.afile.get_select()
```

Selects the ASCII file to be imported or exported.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_extension**(*extension: DexchExtension*) → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:EXTension
driver.source.correction.dexchange.afile.set_extension(extension = enums.
↪DexchExtension.CSV)
```

Determines the extension of the ASCII files for file import or export, or to query existing files.

**param extension**
TXT| CSV

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SELect
driver.source.correction.dexchange.afile.set_select(filename = 'abc')
```

Selects the ASCII file to be imported or exported.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.correction.dexchange.afile.clone()
```

### Subgroups

#### 6.18.6.2.1.1 Separator

### SCPI Commands :

```
[SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:COLumn
[SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:DECimal
```

**class SeparatorCls**

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

**get_column**() → DexchSepCol

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:COLumn
value: enums.DexchSepCol = driver.source.correction.dexchange.afile.separator.
↪get_column()
```

Selects the separator between the frequency and level column of the ASCII table.

**return**
column: TABulator| SEMicolon| COMMa| SPACe

**get_decimal**() → DecimalSeparator

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:DECimal
value: enums.DecimalSeparator = driver.source.correction.dexchange.afile.
↪separator.get_decimal()
```

Sets the decimal separator used in the ASCII data between '.' (decimal point) and ',' (comma) with floating-point numerals.

**return**
decimal: DOT| COMMa

**set_column**(*column: DexchSepCol*) → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:COLumn
driver.source.correction.dexchange.afile.separator.set_column(column = enums.
↪DexchSepCol.COMMa)
```

Selects the separator between the frequency and level column of the ASCII table.

**param column**
TABulator| SEMicolon| COMMa| SPACe

**set_decimal**(*decimal: DecimalSeparator*) → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:DECimal
driver.source.correction.dexchange.afile.separator.set_decimal(decimal = enums.
↪DecimalSeparator.COMMa)
```

Sets the decimal separator used in the ASCII data between '.' (decimal point) and ',' (comma) with floating-point numerals.

**param decimal**
DOT| COMMa

#### 6.18.6.2.2 Execute

**SCPI Command :**

```
[SOURce<HW>]:CORRection:DEXChange:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:EXECute
driver.source.correction.dexchange.execute.set()
```

Executes the import or export of the selected correction list, according to the previously set transfer direction with command [:SOURce<hw>]:CORRection:DEXChange:MODE.

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

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:EXECute
driver.source.correction.dexchange.execute.set_with_opc()
```

Executes the import or export of the selected correction list, according to the previously set transfer direction with command [:SOURce<hw>]:CORRection:DEXChange:MODE.

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

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

#### 6.18.6.3 Zeroing

**SCPI Command :**

```
[SOURce<HW>]:CORRection:ZERoing:STATe
```

**class ZeroingCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:CORRection:ZERoing:STATe
value: bool = driver.source.correction.zeroing.get_state()
```

Activates the zeroing procedure before filling the user correction data acquired by a sensor.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:CORRection:ZERoing:STATe
driver.source.correction.zeroing.set_state(state = False)
```

Activates the zeroing procedure before filling the user correction data acquired by a sensor.

**param state**
1| ON| 0| OFF

### 6.18.7 Dme

**SCPI Command :**

```
[SOURce<HW>]:DME:LOWemission
```

**class DmeCls**

Dme commands group definition. 9 total commands, 1 Subgroups, 1 group commands

**get_low_emission**() → bool

```
# SCPI: [SOURce<HW>]:DME:LOWemission
value: bool = driver.source.dme.get_low_emission()
```

No command help available

**return**
state: No help available

**set_low_emission**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:LOWemission
driver.source.dme.set_low_emission(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.7.1 Analysis

**class AnalysisCls**

Analysis commands group definition. 8 total commands, 4 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.18.7.1.1 Efficiency

### SCPI Commands :

```
[SOURce<HW>]:DME:ANALysis:EFFiciency:OK
[SOURce<HW>]:DME:ANALysis:EFFiciency:STATe
```

**class EfficiencyCls**

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

**get_ok**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:EFFiciency:OK
value: bool = driver.source.dme.analysis.efficiency.get_ok()
```

No command help available

**return**
state: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:EFFiciency:STATe
value: bool = driver.source.dme.analysis.efficiency.get_state()
```

No command help available

**return**
state: No help available

**set_ok**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:EFFiciency:OK
driver.source.dme.analysis.efficiency.set_ok(state = False)
```

No command help available

**param state**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:EFFiciency:STATe
driver.source.dme.analysis.efficiency.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.18.7.1.2 Power

##### SCPI Commands :

```
[SOURce<HW>]:DME:ANALysis:POWer:OK
[SOURce<HW>]:DME:ANALysis:POWer:STATe
```

**class PowerCls**

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

**get_ok**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:POWer:OK
value: bool = driver.source.dme.analysis.power.get_ok()
```

No command help available

**return**
state: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:POWer:STATe
value: bool = driver.source.dme.analysis.power.get_state()
```

No command help available

**return**
state: No help available

**set_ok**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:POWer:OK
driver.source.dme.analysis.power.set_ok(state = False)
```

No command help available

**param state**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:POWer:STATe
driver.source.dme.analysis.power.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.18.7.1.3 PrRate

##### SCPI Commands :

```
[SOURce<HW>]:DME:ANALysis:PRRate:OK
[SOURce<HW>]:DME:ANALysis:PRRate:STATe
```

**class PrRateCls**

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

**get_ok**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:PRRate:OK
value: bool = driver.source.dme.analysis.prRate.get_ok()
```

No command help available

**return**
state: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:PRRate:STATe
value: bool = driver.source.dme.analysis.prRate.get_state()
```

No command help available

**return**
state: No help available

**set_ok**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:PRRate:OK
driver.source.dme.analysis.prRate.set_ok(state = False)
```

No command help available

**param state**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:PRRate:STATe
driver.source.dme.analysis.prRate.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.18.7.1.4 Time

**SCPI Commands :**

```
[SOURce<HW>]:DME:ANALysis:TIME:OK
[SOURce<HW>]:DME:ANALysis:TIME:STATe
```

**class TimeCls**

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

**get_ok**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:TIME:OK
value: bool = driver.source.dme.analysis.time.get_ok()
```

No command help available

**return**
state: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:DME:ANALysis:TIME:STATe
value: bool = driver.source.dme.analysis.time.get_state()
```

No command help available

**return**
state: No help available

**set_ok**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:TIME:OK
driver.source.dme.analysis.time.set_ok(state = False)
```

No command help available

**param state**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:DME:ANALysis:TIME:STATe
driver.source.dme.analysis.time.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.18.8 Fm<GeneratorIx>

#### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.fm.repcap_generatorIx_get()
driver.source.fm.repcap_generatorIx_set(repcap.GeneratorIx.Nr1)
```

#### SCPI Commands :

```
[SOURce<HW>]:FM:MODE
[SOURce<HW>]:FM:RATio
[SOURce<HW>]:FM:SENSitivity
```

**class FmCls**

Fm commands group definition. 8 total commands, 3 Subgroups, 3 group commands Repeated Capability: GeneratorIx, default value after init: GeneratorIx.Nr1

**get_mode**() → FmMode

```
# SCPI: [SOURce<HW>]:FM:MODE
value: enums.FmMode = driver.source.fm.get_mode()
```

Selects the mode for the frequency modulation.

**return**
mode: HBANdwidth| LNOise HBANdwidth Selects maximum range for modulation bandwidth. LNOise Selects optimized phase noise and spurious characteristics with reduced modulation bandwidth and FM deviation.

**get_ratio**() → float

```
# SCPI: [SOURce<HW>]:FM:RATio
value: float = driver.source.fm.get_ratio()
```

Sets the deviation ratio (path2 to path1) in percent.

**return**
ratio: float Range: 0 to 100

**get_sensitivity**() → float

```
# SCPI: [SOURce<HW>]:FM:SENSitivity
value: float = driver.source.fm.get_sensitivity()
```

Queries the sensitivity of the externally supplied signal for frequency modulation. The sensitivity depends on the set modulation deviation.

**return**
sensitivity: float Sensitivity in Hz/V. It is assigned to the voltage value for full modulation of the input. Range: 0 to max

**set_mode**(*mode: FmMode*) → None

```
# SCPI: [SOURce<HW>]:FM:MODE
driver.source.fm.set_mode(mode = enums.FmMode.HBANdwidth)
```

Selects the mode for the frequency modulation.

**param mode**
HBANdwidth| LNOise HBANdwidth Selects maximum range for modulation bandwidth. LNOise Selects optimized phase noise and spurious characteristics with reduced modulation bandwidth and FM deviation.

**set_ratio**(*ratio: float*) → None

```
# SCPI: [SOURce<HW>]:FM:RATio
driver.source.fm.set_ratio(ratio = 1.0)
```

Sets the deviation ratio (path2 to path1) in percent.

**param ratio**
float Range: 0 to 100

### Cloning the Group

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

### Subgroups

#### 6.18.8.1 Deviation

### SCPI Commands :

```
[SOURce<HW>]:FM:DEViation:MODE
[SOURce<HW>]:FM:DEViation:SUM
[SOURce<HW>]:FM<CH>:[DEViation]
```

### class DeviationCls

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

**get**(*generatorIx=GeneratorIx.Default*) → float

```
# SCPI: [SOURce<HW>]:FM<CH>:[DEViation]
value: float = driver.source.fm.deviation.get(generatorIx = repcap.GeneratorIx.
↪Default)
```

Sets the modulation deviation of the frequency modulation in Hz.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fm')

**return**
deviation: float The maximum deviation depends on the RF frequency and the selected modulation mode (see data sheet) . Range: 0 to max

**get_mode**() → ModulationDevMode

```
# SCPI: [SOURce<HW>]:FM:DEViation:MODE
value: enums.ModulationDevMode = driver.source.fm.deviation.get_mode()
```

Selects the coupling mode. The coupling mode parameter also determines the mode for fixing the total deviation.

**return**
fm_dev_mode: UNCoupled| TOTal| RATio UNCoupled Does not couple the LF signals. The deviation values of both paths are independent. TOTal Couples the deviation of both paths. RATio Couples the deviation ratio of both paths

**get_sum**() → float

```
# SCPI: [SOURce<HW>]:FM:DEViation:SUM
value: float = driver.source.fm.deviation.get_sum()
```

Sets the total deviation of the LF signal when using combined signal sources in frequency modulation.

**return**
fm_dev_sum: float Range: 0 to 40E6

**set**(*deviation: float*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:FM<CH>:[DEViation]
driver.source.fm.deviation.set(deviation = 1.0, generatorIx = repcap.
↪GeneratorIx.Default)
```

Sets the modulation deviation of the frequency modulation in Hz.

**param deviation**
float The maximum deviation depends on the RF frequency and the selected modulation mode (see data sheet) . Range: 0 to max

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fm')

**set_mode**(*fm_dev_mode: ModulationDevMode*) → None

```
# SCPI: [SOURce<HW>]:FM:DEViation:MODE
driver.source.fm.deviation.set_mode(fm_dev_mode = enums.ModulationDevMode.RATio)
```

Selects the coupling mode. The coupling mode parameter also determines the mode for fixing the total deviation.

**param fm_dev_mode**
UNCoupled| TOTal| RATio UNCoupled Does not couple the LF signals. The deviation values of both paths are independent. TOTal Couples the deviation of both paths. RATio Couples the deviation ratio of both paths

**set_sum**(*fm_dev_sum: float*) → None

```
# SCPI: [SOURce<HW>]:FM:DEViation:SUM
driver.source.fm.deviation.set_sum(fm_dev_sum = 1.0)
```

Sets the total deviation of the LF signal when using combined signal sources in frequency modulation.

**param fm_dev_sum**
float Range: 0 to 40E6

#### 6.18.8.2 Source

**SCPI Command :**

```
[SOURce<HW>]:FM<CH>:SOURce
```

**class SourceCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → FmSour

```
# SCPI: [SOURce<HW>]:FM<CH>:SOURce
value: enums.FmSour = driver.source.fm.source.get(generatorIx = repcap.
↪GeneratorIx.Default)
```

Selects the modulation source for frequency modulation.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fm')

**return**
source: LF1| LF2| NOISe| EXT1| INTernal| EXTernal | EXT2 LF1|LF2 Uses an internally generated LF signal. INTernal = LF1 Works like LF1 EXTernal Works like EXT1 EXT1|EXT2 Uses an externally supplied LF signal. NOISe Uses the internally generated noise signal.

**set**(*source: FmSour*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:FM<CH>:SOURce
driver.source.fm.source.set(source = enums.FmSour.EXT1, generatorIx = repcap.
↪GeneratorIx.Default)
```

Selects the modulation source for frequency modulation.

**param source**
LF1| LF2| NOISe| EXT1| INTernal| EXTernal | EXT2 LF1|LF2 Uses an internally generated LF signal. INTernal = LF1 Works like LF1 EXTernal Works like EXT1 EXT1|EXT2 Uses an externally supplied LF signal. NOISe Uses the internally generated noise signal.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fm')

#### 6.18.8.3 State

**SCPI Command :**

```
[SOURce<HW>]:FM<CH>:STATe
```

`class StateCls`

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

`get`(*generatorIx*=*GeneratorIx.Default*) → bool

```
# SCPI: [SOURce<HW>]:FM<CH>:STATe
value: bool = driver.source.fm.state.get(generatorIx = repcap.GeneratorIx.
↪Default)
```

Activates frequency modulation.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fm')

**return**
state: 1| ON| 0| OFF

`set`(*state: bool*, *generatorIx*=*GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:FM<CH>:STATe
driver.source.fm.state.set(state = False, generatorIx = repcap.GeneratorIx.
↪Default)
```

Activates frequency modulation.

**param state**
1| ON| 0| OFF

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fm')

### 6.18.9 FreqSweep

**class FreqSweepCls**

FreqSweep 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.source.freqSweep.clone()
```

#### Subgroups

##### 6.18.9.1 Trigger

**class TriggerCls**

Trigger 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.source.freqSweep.trigger.clone()
```

#### Subgroups

##### 6.18.9.1.1 Source

#### SCPI Command :

```
[SOURce<HW>]:FSWeep:TRIGger:SOURce:ADVanced
```

**class SourceCls**

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

**get_advanced**() → TrigSweepImmBusExt

```
# SCPI: [SOURce<HW>]:FSWeep:TRIGger:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.source.freqSweep.trigger.source.get_
↪advanced()
```

No command help available

**return**
imm_bus_ext: No help available

**set_advanced**(*imm_bus_ext: TrigSweepImmBusExt*) → None

```
# SCPI: [SOURce<HW>]:FSWeep:TRIGger:SOURce:ADVanced
driver.source.freqSweep.trigger.source.set_advanced(imm_bus_ext = enums.
↪TrigSweepImmBusExt.BUS)
```

No command help available

**param imm_bus_ext**
No help available

### 6.18.10 Frequency

#### SCPI Commands :

```
[SOURce<HW>]:FREQuency:CENTer
[SOURce<HW>]:FREQuency:FREQuency
[SOURce<HW>]:FREQuency:MANual
[SOURce<HW>]:FREQuency:MODE
[SOURce<HW>]:FREQuency:OFFSet
[SOURce<HW>]:FREQuency:SPAN
[SOURce<HW>]:FREQuency:STARt
[SOURce<HW>]:FREQuency:STOP
```

**class FrequencyCls**

Frequency commands group definition. 50 total commands, 6 Subgroups, 8 group commands

**get_center**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:CENTer
value: float = driver.source.frequency.get_center()
```

Sets the center frequency of the sweep. See 'Correlating parameters in sweep mode'.

**return**
center: float Range: 300 kHz to RFmax, Unit: Hz

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:FREQuency
value: float = driver.source.frequency.get_frequency()
```

No command help available

**return**
frequency: No help available

**get_manual**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MANual
value: float = driver.source.frequency.get_manual()
```

Sets the frequency and triggers a sweep step manually if SWEep:MODE MAN.

**return**
manual: float You can select any frequency within the setting range, where: STARt is set with [:SOURcehw]:FREQuency:STARt STOP is set with [:SOURcehw]:FREQuency:STOP OFFSet is set with [:SOURcehw]:FREQuency:OFFSet Range: (STARt + OFFSet) to (STOP + OFFSet) , Unit: Hz

**get_mode**() → FreqMode

```
# SCPI: [SOURce<HW>]:FREQuency:MODE
value: enums.FreqMode = driver.source.frequency.get_mode()
```

Sets the frequency mode for generating the RF output signal. The selected mode determines the parameters to be used for further frequency settings.

**return**
mode: CW| FIXed | SWEep| LIST | COMBined CW|FIXed Sets the fixed frequency mode. CW and FIXed are synonyms. The instrument operates at a defined frequency, set with command [:SOURcehw]:FREQuency[:CW|FIXed]. SWEep Sets sweep mode. The instrument processes frequency (and level) settings in defined sweep steps. Set the range and current frequency with the commands: [:SOURcehw]:FREQuency:STARt and [:SOURcehw]:FREQuency:STOP, [:SOURcehw]:FREQuency:CENTer, [:SOURcehw]:FREQuency:SPAN, [:SOURcehw]:FREQuency:MANual LIST Sets list mode. The instrument processes frequency and level settings by means of values loaded from a list. To configure list mode settings, use the commands of the 'SOURce:LIST subsystem'. COMBined Sets the combined RF frequency / level sweep mode. The instrument processes frequency and level settings in defined sweep steps. Set the range and current frequency with the commands: [:SOURcehw]:COMBined:FREQuency:STARt and [:SOURcehw]:COMBined:FREQuency:STOP, [:SOURcehw]:COMBined:POWer:STARt and [:SOURcehw]:COMBined:POWer:STOP

**get_offset**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:OFFSet
value: float = driver.source.frequency.get_offset()
```

Sets the frequency offset fFREQ:OFFSet of a downstream instrument. The parameters offset fFREQ:OFFSer and multiplier NFREQ:MULT affect the frequency value set with the command [:SOURce<hw>]:FREQuency[:CW|FIXed]. The query [:SOURce<hw>]:FREQuency[:CW|FIXed] returns the value corresponding to the formula: fFREQ = fRFout * NFREQ:MULT + fFREQ:OFFSer See 'RF frequency and level display with a downstream instrument'. Note: The offset also affects RF frequency sweep.

**return**
offset: float

**get_span**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:SPAN
value: float = driver.source.frequency.get_span()
```

Sets the sapn of the frequency sweep range. See 'Correlating parameters in sweep mode'.

**return**
span: float Full freqeuncy range

**get_start**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:STARt
value: float = driver.source.frequency.get_start()
```

Sets the start frequency for the RF sweep. See 'Correlating parameters in sweep mode'.

**return**
start: float Range: 300kHz to RFmax

**get_stop**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:STOP
value: float = driver.source.frequency.get_stop()
```

Sets the stop frequency range for the RF sweep. See 'Correlating parameters in sweep mode'.

**return**
stop: float Range: 300kHz to RFmax, Unit: Hz

**set_center**(*center: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:CENTer
driver.source.frequency.set_center(center = 1.0)
```

Sets the center frequency of the sweep. See 'Correlating parameters in sweep mode'.

**param center**
float Range: 300 kHz to RFmax, Unit: Hz

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:FREQuency
driver.source.frequency.set_frequency(frequency = 1.0)
```

No command help available

**param frequency**
No help available

**set_manual**(*manual: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MANual
driver.source.frequency.set_manual(manual = 1.0)
```

Sets the frequency and triggers a sweep step manually if SWEep:MODE MAN.

**param manual**
float You can select any frequency within the setting range, where: STARt is set with [:SOURcehw]:FREQuency:STARt STOP is set with [:SOURcehw]:FREQuency:STOP OFFSet is set with [:SOURcehw]:FREQuency:OFFSet Range: (STARt + OFFSet) to (STOP + OFFSet) , Unit: Hz

**set_mode**(*mode: FreqMode*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MODE
driver.source.frequency.set_mode(mode = enums.FreqMode.COMBined)
```

Sets the frequency mode for generating the RF output signal. The selected mode determines the parameters to be used for further frequency settings.

**param mode**
CW| FIXed | SWEep| LIST | COMBined CW|FIXed Sets the fixed frequency mode. CW and FIXed are synonyms. The instrument operates at a defined frequency, set with command [:SOURcehw]:FREQuency[:CW|FIXed]. SWEep Sets sweep mode. The

instrument processes frequency (and level) settings in defined sweep steps. Set the range and current frequency with the commands: [:SOURcehw]:FREQuency:STARt and [:SOURcehw]:FREQuency:STOP, [:SOURcehw]:FREQuency:CENTer, [:SOURcehw]:FREQuency:SPAN, [:SOURcehw]:FREQuency:MANual LIST Sets list mode. The instrument processes frequency and level settings by means of values loaded from a list. To configure list mode settings, use the commands of the 'SOURce:LIST subsystem'. COMBined Sets the combined RF frequency / level sweep mode. The instrument processes frequency and level settings in defined sweep steps. Set the range and current frequency with the commands: [:SOURcehw]:COMBined:FREQuency:STARt and [:SOURcehw]:COMBined:FREQuency:STOP, [:SOURcehw]:COMBined:POWer:STARt and [:SOURcehw]:COMBined:POWer:STOP

**set_offset**(*offset: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:OFFSet
driver.source.frequency.set_offset(offset = 1.0)
```

Sets the frequency offset fFREQ:OFFSet of a downstream instrument. The parameters offset fFREQ:OFFSer and multiplier NFREQ:MULT affect the frequency value set with the command [:SOURce<hw>]:FREQuency[:CW|FIXed]. The query [:SOURce<hw>]:FREQuency[:CW|FIXed] returns the value corresponding to the formula: fFREQ = fRFout * NFREQ:MULT + fFREQ:OFFSer See 'RF frequency and level display with a downstream instrument'. Note: The offset also affects RF frequency sweep.

**param offset**
float

**set_span**(*span: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:SPAN
driver.source.frequency.set_span(span = 1.0)
```

Sets the sapn of the frequency sweep range. See 'Correlating parameters in sweep mode'.

**param span**
float Full freqeuncy range

**set_start**(*start: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:STARt
driver.source.frequency.set_start(start = 1.0)
```

Sets the start frequency for the RF sweep. See 'Correlating parameters in sweep mode'.

**param start**
float Range: 300kHz to RFmax

**set_stop**(*stop: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:STOP
driver.source.frequency.set_stop(stop = 1.0)
```

Sets the stop frequency range for the RF sweep. See 'Correlating parameters in sweep mode'.

**param stop**
float Range: 300kHz to RFmax, Unit: Hz

### Cloning the Group

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

### Subgroups

#### 6.18.10.1 Cw

### SCPI Commands :

```
[SOURce<HW>]:FREQuency:[CW]:RCL
[SOURce<HW>]:FREQuency:[CW]
```

**class CwCls**

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

**get_recall**() → InclExcl

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:RCL
value: enums.InclExcl = driver.source.frequency.cw.get_recall()
```

Set whether the RF frequency value is retained or taken from a loaded instrument configuration, when you recall instrument settings with command *RCL.

**return**

rcl: INCLude| EXCLude INCLude Takes the frequency value of the loaded settings. EXCLude Retains the current frequency when an instrument configuration is loaded.

**get_value**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]
value: float = driver.source.frequency.cw.get_value()
```

**Sets the frequency of the RF output signal in the selected path.**

INTRO_CMD_HELP: The effect depends on the selected mode:

- In CW mode (FREQ:MODE CW | FIXed) , the instrument operates at a fixed frequency.
- In sweep mode (FREQ:MODE SWE) , the value applies to the sweep frequency. The instrument processes the frequency settings in defined sweep steps.
- In user mode (FREQ:STEP:MODE USER) , you can vary the current frequency step by step.

**return**

fixed: float The following settings influence the value range: An offset set with the command [:SOURcehw]:FREQuency:OFFSet Numerical value Sets the frequency in CW and sweep mode UP|DOWN Varies the frequency step by step in user mode. The frequency is increased or decreased by the value set with the command [:SOURcehw]:FREQuency:STEP[:INCRement]. Range: (RFmin + OFFSet) to (RFmax + OFFSet)

**set_recall**(*rcl: InclExcl*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:RCL
driver.source.frequency.cw.set_recall(rcl = enums.InclExcl.EXCLude)
```

Set whether the RF frequency value is retained or taken from a loaded instrument configuration, when you recall instrument settings with command *RCL.

**param rcl**
INCLude| EXCLude INCLude Takes the frequency value of the loaded settings. EXCLude Retains the current frequency when an instrument configuration is loaded.

**set_value**(*fixed: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]
driver.source.frequency.cw.set_value(fixed = 1.0)
```

**Sets the frequency of the RF output signal in the selected path.**
INTRO_CMD_HELP: The effect depends on the selected mode:

- In CW mode (FREQ:MODE CW | FIXed) , the instrument operates at a fixed frequency.
- In sweep mode (FREQ:MODE SWE) , the value applies to the sweep frequency. The instrument processes the frequency settings in defined sweep steps.
- In user mode (FREQ:STEP:MODE USER) , you can vary the current frequency step by step.

**param fixed**
float The following settings influence the value range: An offset set with the command [:SOURcehw]:FREQuency:OFFSet Numerical value Sets the frequency in CW and sweep mode UP|DOWN Varies the frequency step by step in user mode. The frequency is increased or decreased by the value set with the command [:SOURcehw]:FREQuency:STEP[:INCRement]. Range: (RFmin + OFFSet) to (RFmax + OFFSet)

#### 6.18.10.2 Fixed

**SCPI Commands :**

```
[SOURce<HW>]:FREQuency:[FIXed]:RCL
[SOURce<HW>]:FREQuency:[FIXed]
```

**class FixedCls**

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

**get_recall**() → InclExcl

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:RCL
value: enums.InclExcl = driver.source.frequency.fixed.get_recall()
```

Set whether the RF frequency value is retained or taken from a loaded instrument configuration, when you recall instrument settings with command *RCL.

**return**
rcl: INCLude| EXCLude INCLude Takes the frequency value of the loaded settings. EXCLude Retains the current frequency when an instrument configuration is loaded.

**get_value**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]
value: float = driver.source.frequency.fixed.get_value()
```

**Sets the frequency of the RF output signal in the selected path.**
INTRO_CMD_HELP: The effect depends on the selected mode:

- In CW mode (FREQ:MODE CW | FIXed) , the instrument operates at a fixed frequency.
- In sweep mode (FREQ:MODE SWE) , the value applies to the sweep frequency. The instrument processes the frequency settings in defined sweep steps.
- In user mode (FREQ:STEP:MODE USER) , you can vary the current frequency step by step.

**return**
fixed: float The following settings influence the value range: An offset set with the command [:SOURcehw]:FREQuency:OFFSet Numerical value Sets the frequency in CW and sweep mode UP|DOWN Varies the frequency step by step in user mode. The frequency is increased or decreased by the value set with the command [:SOURcehw]:FREQuency:STEP[:INCRement]. Range: (RFmin + OFFSet) to (RFmax + OFFSet)

**set_recall**(*rcl: InclExcl*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:RCL
driver.source.frequency.fixed.set_recall(rcl = enums.InclExcl.EXCLude)
```

Set whether the RF frequency value is retained or taken from a loaded instrument configuration, when you recall instrument settings with command *RCL.

**param rcl**
INCLude| EXCLude INCLude Takes the frequency value of the loaded settings. EXCLude Retains the current frequency when an instrument configuration is loaded.

**set_value**(*fixed: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]
driver.source.frequency.fixed.set_value(fixed = 1.0)
```

**Sets the frequency of the RF output signal in the selected path.**
INTRO_CMD_HELP: The effect depends on the selected mode:

- In CW mode (FREQ:MODE CW | FIXed) , the instrument operates at a fixed frequency.
- In sweep mode (FREQ:MODE SWE) , the value applies to the sweep frequency. The instrument processes the frequency settings in defined sweep steps.
- In user mode (FREQ:STEP:MODE USER) , you can vary the current frequency step by step.

**param fixed**
float The following settings influence the value range: An offset set with the command [:SOURcehw]:FREQuency:OFFSet Numerical value Sets the frequency in CW and sweep mode UP|DOWN Varies the frequency step by step in user mode. The frequency is increased or decreased by the value set with the command [:SOURcehw]:FREQuency:STEP[:INCRement]. Range: (RFmin + OFFSet) to (RFmax + OFFSet)

#### 6.18.10.3 Multiplier

**SCPI Command :**

```
[SOURce<HW>]:FREQuency:MULTiplier
```

**class MultiplierCls**

Multiplier commands group definition. 31 total commands, 1 Subgroups, 1 group commands

**get_value**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier
value: float = driver.source.frequency.multiplier.get_value()
```

Sets the multiplication factor NFREQ:MULT of a subsequent downstream instrument. The parameters offset fFREQ:OFFSer and multiplier NFREQ:MULT affect the frequency value set with the command [:SOURce<hw>]:FREQuency[:CW|FIXed]. The query [:SOURce<hw>]:FREQuency[:CW|FIXed] returns the value corresponding to the formula: fFREQ = fRFout * NFREQ:MULT + fFREQ:OFFSer See 'RF frequency and level display with a downstream instrument'.

**return**
multiplier: float Range: -10000 to 10000

**set_value**(*multiplier: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier
driver.source.frequency.multiplier.set_value(multiplier = 1.0)
```

Sets the multiplication factor NFREQ:MULT of a subsequent downstream instrument. The parameters offset fFREQ:OFFSer and multiplier NFREQ:MULT affect the frequency value set with the command [:SOURce<hw>]:FREQuency[:CW|FIXed]. The query [:SOURce<hw>]:FREQuency[:CW|FIXed] returns the value corresponding to the formula: fFREQ = fRFout * NFREQ:MULT + fFREQ:OFFSer See 'RF frequency and level display with a downstream instrument'.

**param multiplier**
float Range: -10000 to 10000

**Cloning the Group**

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

**Subgroups**

##### 6.18.10.3.1 External

**SCPI Commands :**

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:DAC0
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:DAC1
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FMAXimum
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FMINimum
```

(continues on next page)

(continued from previous page)

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:IPMax
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:IPOWer
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:MULTiplier
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PADJust
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PMAXimum
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PMINimum
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PSDMinimum
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:REVision
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:SNUMber
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:STATe
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:TYPE
```

**class ExternalCls**

External commands group definition. 30 total commands, 3 Subgroups, 15 group commands

**get_dac_0**() → int

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:DAC0
value: int = driver.source.frequency.multiplier.external.get_dac_0()
```

No command help available

**return**
dac_0_value: No help available

**get_dac_1**() → int

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:DAC1
value: int = driver.source.frequency.multiplier.external.get_dac_1()
```

No command help available

**return**
dac_1_value: No help available

**get_fmaximum**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FMAXimum
value: float = driver.source.frequency.multiplier.external.get_fmaximum()
```

No command help available

**return**
fmax: No help available

**get_fminimum**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FMINimum
value: float = driver.source.frequency.multiplier.external.get_fminimum()
```

No command help available

**return**
fmin: No help available

**get_ipmax**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:IPMax
value: float = driver.source.frequency.multiplier.external.get_ipmax()
```

No command help available

**return**
input_power_max: No help available

**get_ipower**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:IPOWer
value: float = driver.source.frequency.multiplier.external.get_ipower()
```

No command help available

**return**
input_power: No help available

**get_multiplier**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:MULTiplier
value: float = driver.source.frequency.multiplier.external.get_multiplier()
```

No command help available

**return**
multiplier: No help available

**get_padjust**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PADJust
value: float = driver.source.frequency.multiplier.external.get_padjust()
```

No command help available

**return**
power_adjust: No help available

**get_pmaximum**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PMAXimum
value: float = driver.source.frequency.multiplier.external.get_pmaximum()
```

No command help available

**return**
pmax: No help available

**get_pminimum**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PMINimum
value: float = driver.source.frequency.multiplier.external.get_pminimum()
```

No command help available

**return**
pmin: No help available

**get_psd_minimum**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:PSDMinimum
value: float = driver.source.frequency.multiplier.external.get_psd_minimum()
```

No command help available

**return**
power_sweep_dwell: No help available

**get_revision**() → str

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:REVision
value: str = driver.source.frequency.multiplier.external.get_revision()
```

No command help available

**return**
revision: No help available

**get_snumber**() → str

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:SNUMber
value: str = driver.source.frequency.multiplier.external.get_snumber()
```

No command help available

**return**
serial_number: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:STATe
value: bool = driver.source.frequency.multiplier.external.get_state()
```

No command help available

**return**
state: No help available

**get_type_py**() → str

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:TYPE
value: str = driver.source.frequency.multiplier.external.get_type_py()
```

No command help available

**return**
type_py: No help available

**set_dac_0**(*dac_0_value: int*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:DAC0
driver.source.frequency.multiplier.external.set_dac_0(dac_0_value = 1)
```

No command help available

**param dac_0_value**
No help available

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.frequency.multiplier.external.clone()
```

### Subgroups

#### 6.18.10.3.1.1 Correction

### SCPI Commands :

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:CATalog
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:CLOSs
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:DELete
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:MODE
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:SELect
```

**class CorrectionCls**

Correction commands group definition. 10 total commands, 3 Subgroups, 5 group commands

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:DELete
driver.source.frequency.multiplier.external.correction.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:CATalog
value: List[str] = driver.source.frequency.multiplier.external.correction.get_
↪catalog()
```

No command help available

**return**
catalog: No help available

**get_closs**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:CLOSs
value: float = driver.source.frequency.multiplier.external.correction.get_
↪closs()
```

No command help available

**return**
cable_loss: No help available

**get_mode**() → RfFreqMultCcorMode

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:MODE
value: enums.RfFreqMultCcorMode = driver.source.frequency.multiplier.external.
↪correction.get_mode()
```

No command help available

**return**
mode: No help available

**get_select**() → str

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:SELect
value: str = driver.source.frequency.multiplier.external.correction.get_select()
```

No command help available

**return**
filename: No help available

**set_closs**(*cable_loss: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:CLOSs
driver.source.frequency.multiplier.external.correction.set_closs(cable_loss = 1.
↪0)
```

No command help available

**param cable_loss**
No help available

**set_mode**(*mode: RfFreqMultCcorMode*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:MODE
driver.source.frequency.multiplier.external.correction.set_mode(mode = enums.
↪RfFreqMultCcorMode.HPRecision)
```

No command help available

**param mode**
No help available

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:SELect
driver.source.frequency.multiplier.external.correction.set_select(filename =
↪'abc')
```

No command help available

**param filename**
No help available

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.frequency.multiplier.external.correction.clone()
```

#### Subgroups

##### 6.18.10.3.1.2 Frequency

#### SCPI Commands :

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:FREQuency:POINts
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:FREQuency
```

**class FrequencyCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:FREQuency:POINts
value: int = driver.source.frequency.multiplier.external.correction.frequency.
↪get_points()
```

No command help available

**return**
points: No help available

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:FREQuency
value: List[float] = driver.source.frequency.multiplier.external.correction.
↪frequency.get_value()
```

No command help available

**return**
list_freq: No help available

**set_value**(*list_freq: List[float]*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:FREQuency
driver.source.frequency.multiplier.external.correction.frequency.set_value(list_
↪freq = [1.1, 2.2, 3.3])
```

No command help available

**param list_freq**
No help available

#### 6.18.10.3.1.3 Power

##### SCPI Commands :

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:POWer:POINts
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:POWer
```

**class PowerCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:POWer:POINts
value: int = driver.source.frequency.multiplier.external.correction.power.get_
↪points()
```

No command help available

**return**
points: No help available

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:POWer
value: List[float] = driver.source.frequency.multiplier.external.correction.
↪power.get_value()
```

No command help available

**return**
list_pow: No help available

**set_value**(*list_pow: List[float]*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:POWer
driver.source.frequency.multiplier.external.correction.power.set_value(list_pow␣
↪= [1.1, 2.2, 3.3])
```

No command help available

**param list_pow**
No help available

#### 6.18.10.3.1.4 Sensor<Channel>

##### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.frequency.multiplier.external.correction.sensor.repcap_channel_get()
driver.source.frequency.multiplier.external.correction.sensor.repcap_channel_set(repcap.
↪Channel.Nr1)
```

**class SensorCls**

Sensor commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.frequency.multiplier.external.correction.sensor.clone()
```

#### Subgroups

##### 6.18.10.3.1.5 Sonce

#### SCPI Command :

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:SENSor<CH>:SONCe
```

**class SonceCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:CORRection:SENSor<CH>:SONCe
driver.source.frequency.multiplier.external.correction.sensor.sonce.set(channel␣
↪= repcap.Channel.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

##### 6.18.10.3.1.6 Firmware

#### SCPI Commands :

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:CATalog
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:SELect
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:VERSion
```

**class FirmwareCls**

Firmware commands group definition. 4 total commands, 1 Subgroups, 3 group commands

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:CATalog
value: List[str] = driver.source.frequency.multiplier.external.firmware.get_
↪catalog()
```

No command help available

**return**
catalog: No help available

**get_select**() → str

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:SELect
value: str = driver.source.frequency.multiplier.external.firmware.get_select()
```

No command help available

**return**
filename: No help available

**get_version**() → str

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:VERSion
value: str = driver.source.frequency.multiplier.external.firmware.get_version()
```

No command help available

**return**
firmware_version: No help available

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:SELect
driver.source.frequency.multiplier.external.firmware.set_select(filename = 'abc
↪')
```

No command help available

**param filename**
No help available

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.frequency.multiplier.external.firmware.clone()
```

### Subgroups

#### 6.18.10.3.1.7 Update

### SCPI Command :

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:UPDate
```

**class UpdateCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:UPDate
driver.source.frequency.multiplier.external.firmware.update.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:FIRMware:UPDate
driver.source.frequency.multiplier.external.firmware.update.set_with_opc()
```

No command help available

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

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

#### 6.18.10.3.1.8 Loader

##### SCPI Command :

```
[SOURce<HW>]:FREQuency:MULTiplier:EXTernal:LOADer:VERSion
```

**class LoaderCls**

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

**get_version**() → str

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier:EXTernal:LOADer:VERSion
value: str = driver.source.frequency.multiplier.external.loader.get_version()
```

No command help available

**return**
loader_version: No help available

### 6.18.10.4 Phase

**class PhaseCls**

Phase 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.source.frequency.phase.clone()
```

#### Subgroups

#### 6.18.10.4.1 Continuous

##### SCPI Commands :

```
[SOURce<HW>]:FREQuency:PHASe:CONTinuous:HIGH
[SOURce<HW>]:FREQuency:PHASe:CONTinuous:LOW
[SOURce<HW>]:FREQuency:PHASe:CONTinuous:MODE
[SOURce<HW>]:FREQuency:PHASe:CONTinuous:STATe
```

**class ContinuousCls**

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

**get_high**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:PHASe:CONTinuous:HIGH
value: float = driver.source.frequency.phase.continuous.get_high()
```

Queries the maximum frequency of the frequency range for phase continuous settings. The maximum frequency of the frequency range depends on the mode selected with the command [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:MODE.

**return**
high: float Range: 1E5 to 6E9, Unit: Hz

**get_low**() → float

```
# SCPI: [SOURce<HW>]:FREQuency:PHASe:CONTinuous:LOW
value: float = driver.source.frequency.phase.continuous.get_low()
```

Queries the minimum frequency of the frequency range for phase continuous settings. The minimum frequency of the frequency range depends on the mode selected with the command [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:MODE.

**return**
low: float Range: 1E5 to 6E9, Unit: Hz

**get_mode**() → FilterWidth

```
# SCPI: [SOURce<HW>]:FREQuency:PHASe:CONTinuous:MODE
value: enums.FilterWidth = driver.source.frequency.phase.continuous.get_mode()
```

Selects the mode that determines the frequency range for the phase continuity. To query the frequency range, use the commands [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:HIGH? and [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:LOW?

**return**
mode: NARRow| WIDE NARRow Small frequency range, asymmetrically around the RF frequency. WIDE Large frequency range, symmetrically around the RF frequency.

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:FREQuency:PHASe:CONTinuous:STATe
value: bool = driver.source.frequency.phase.continuous.get_state()
```

Activates phase continuity of the RF frequency. The frequency range is limited and varies depending on the set RF frequency. You can query the range with the commands [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:HIGH? and [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:LOW?. Note: Restricted structure of command line. In phase continuous mode, the R&S SMA100B only processes the first command of a command line and ignores further commands if they are on the same line.

**return**
state: 1| ON| 0| OFF

**set_mode**(*mode: FilterWidth*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:PHASe:CONTinuous:MODE
driver.source.frequency.phase.continuous.set_mode(mode = enums.FilterWidth.
↪NARRow)
```

Selects the mode that determines the frequency range for the phase continuity. To query the frequency range, use the commands [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:HIGH? and [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:LOW?

**param mode**
NARRow| WIDE NARRow Small frequency range, asymmetrically around the RF frequency. WIDE Large frequency range, symmetrically around the RF frequency.

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:PHASe:CONTinuous:STATe
driver.source.frequency.phase.continuous.set_state(state = False)
```

Activates phase continuity of the RF frequency. The frequency range is limited and varies depending on the set RF frequency. You can query the range with the commands [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:HIGH? and [:SOURce<hw>]:FREQuency:PHASe:CONTinuous:LOW?. Note: Restricted structure of command line. In phase continuous mode, the R&S SMA100B only processes the first command of a command line and ignores further commands if they are on the same line.

**param state**
1| ON| 0| OFF

#### 6.18.10.5 Pll

##### SCPI Command :

```
[SOURce<HW>]:FREQuency:PLL:MODE
```

**class PllCls**

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

**get_mode**() → FreqPllModeF

```
# SCPI: [SOURce<HW>]:FREQuency:PLL:MODE
value: enums.FreqPllModeF = driver.source.frequency.pll.get_mode()
```

Selects the PLL (Phase Locked Loop) bandwidth of the main synthesizer.

**return**
mode: NORMal| NARRow NORMal Maximum modulation bandwidth and FM/PhiM deviation. NARRow Narrow PLL bandwidth

**set_mode**(*mode: FreqPllModeF*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:PLL:MODE
driver.source.frequency.pll.set_mode(mode = enums.FreqPllModeF.NARRow)
```

Selects the PLL (Phase Locked Loop) bandwidth of the main synthesizer.

**param mode**
NORMal| NARRow NORMal Maximum modulation bandwidth and FM/PhiM deviation. NARRow Narrow PLL bandwidth

#### 6.18.10.6 Step

**SCPI Commands :**

```
[SOURce<HW>]:FREQuency:STEP:MODE
[SOURce<HW>]:FREQuency:STEP:[INCRement]
```

`class StepCls`

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

`get_increment()` → float

```
# SCPI: [SOURce<HW>]:FREQuency:STEP:[INCRement]
value: float = driver.source.frequency.step.get_increment()
```

Sets the step width. You can use this value to vary the RF frequency with command FREQ UP or FREQ DOWN, if you have activated FREQ:STEP:MODE USER. Note: This value also applies to the step width of the rotary knob on the instrument and, in user-defined step mode, increases or decreases the frequency.

**return**
increment: float Range: 0 Hz to RFmax - 100 kHz

`get_mode()` → FreqStepMode

```
# SCPI: [SOURce<HW>]:FREQuency:STEP:MODE
value: enums.FreqStepMode = driver.source.frequency.step.get_mode()
```

Defines the type of step size to vary the RF frequency at discrete steps with the commands FREQ UP or FREQ DOWN.

**return**
mode: DECimal| USER DECimal Increases or decreases the level in steps of ten. USER Increases or decreases the level in increments, set with the command FREQ:STEP[:INCR].

`set_increment(increment: float)` → None

```
# SCPI: [SOURce<HW>]:FREQuency:STEP:[INCRement]
driver.source.frequency.step.set_increment(increment = 1.0)
```

Sets the step width. You can use this value to vary the RF frequency with command FREQ UP or FREQ DOWN, if you have activated FREQ:STEP:MODE USER. Note: This value also applies to the step width of the rotary knob on the instrument and, in user-defined step mode, increases or decreases the frequency.

**param increment**
float Range: 0 Hz to RFmax - 100 kHz

`set_mode(mode: FreqStepMode)` → None

```
# SCPI: [SOURce<HW>]:FREQuency:STEP:MODE
driver.source.frequency.step.set_mode(mode = enums.FreqStepMode.DECimal)
```

Defines the type of step size to vary the RF frequency at discrete steps with the commands FREQ UP or FREQ DOWN.

**param mode**
DECimal| USER DECimal Increases or decreases the level in steps of ten. USER Increases or decreases the level in increments, set with the command FREQ:STEP[:INCR].

### 6.18.11 Ils

#### SCPI Commands :

```
[SOURce<HW>]:ILS:PRESet
[SOURce<HW>]:ILS:STATe
[SOURce<HW>]:ILS:TYPE
```

**class IlsCls**

Ils commands group definition. 96 total commands, 5 Subgroups, 3 group commands

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:ILS:STATe
value: bool = driver.source.ils.get_state()
```

Activates/deactivates the VOR modulation.

**return**
state: 1| ON| 0| OFF

**get_type_py**() → AvionicIlsType

```
# SCPI: [SOURce<HW>]:ILS:TYPE
value: enums.AvionicIlsType = driver.source.ils.get_type_py()
```

Selects the ILS modulation type.

**return**
type_py: GS| LOCalize| GSLope| MBEacon

**preset**() → None

```
# SCPI: [SOURce<HW>]:ILS:PRESet
driver.source.ils.preset()
```

Sets the parameters of the digital standard to their default values (*RST values specified for the commands) . Not affected is the state set with the command SOURce<hw>:VOR:STATe.

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

```
# SCPI: [SOURce<HW>]:ILS:PRESet
driver.source.ils.preset_with_opc()
```

Sets the parameters of the digital standard to their default values (*RST values specified for the commands) . Not affected is the state set with the command SOURce<hw>:VOR:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:ILS:STATe
driver.source.ils.set_state(state = False)
```

Activates/deactivates the VOR modulation.

**param state**
1| ON| 0| OFF

**set_type_py**(*type_py: AvionicIlsType*) → None

```
# SCPI: [SOURce<HW>]:ILS:TYPE
driver.source.ils.set_type_py(type_py = enums.AvionicIlsType.GS)
```

Selects the ILS modulation type.

**param type_py**
GS| LOCalize| GSLope| MBEacon

### Cloning the Group

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

### Subgroups

#### 6.18.11.1 Gs

### SCPI Commands :

```
[SOURce<HW>]:ILS:GS:PRESet
[SOURce<HW>]:ILS:GS:STATe
[SOURce<HW>]:ILS:[GS]:MODE
[SOURce<HW>]:ILS:[GS]:PHASe
[SOURce<HW>]:ILS:[GS]:SDM
[SOURce<HW>]:ILS:[GS]:SOURce
```

**class GsCls**

Gs commands group definition. 20 total commands, 5 Subgroups, 6 group commands

**get_mode**() → AvionicIlsGsMode

```
# SCPI: [SOURce<HW>]:ILS:[GS]:MODE
value: enums.AvionicIlsGsMode = driver.source.ils.gs.get_mode()
```

Sets the operating mode for the ILS glide slope modulation signal.

**return**
mode: NORM| ULOBe| LLOBe NORM ILS glide slope modulation is active. ULOBe Amplitude modulation of the output signal with the upper lobe (90Hz) signal component of the ILS glide slope signal is active. LLOBe Amplitude modulation of the output signal with the lower lobe (150Hz) signal component of the ILS glide slope signal is active.

**get_phase**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:PHASe
value: float = driver.source.ils.gs.get_phase()
```

Sets the phase between the modulation signals of the upper and lower antenna lobe of the ILS glide slope signal. Zero crossing of the lower lobe (150Hz) signal serves as a reference. The angle refers to the period of the signal of the right antenna lobe.

**return**
phase: float Range: -60 to 120

**get_sdm**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:SDM
value: float = driver.source.ils.gs.get_sdm()
```

Sets the arithmetic sum of the modulation depths of the upper lobe (90 Hz) and lower lobe (150 Hz) for the ILS glide slope signal contents. The RMS modulation depth of the sum signal depends on the phase setting of both modulation tones.

**return**
sdm: float Range: 0 to 100

**get_source**() → AvionicExtAm

```
# SCPI: [SOURce<HW>]:ILS:[GS]:SOURce
value: enums.AvionicExtAm = driver.source.ils.gs.get_source()
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**return**
ils_gs_source: INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:ILS:GS:STATe
value: bool = driver.source.ils.gs.get_state()
```

No command help available

**return**
state: No help available

**preset**() → None

```
# SCPI: [SOURce<HW>]:ILS:GS:PRESet
driver.source.ils.gs.preset()
```

Sets the parameters of the ILS glide slope component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

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

```
# SCPI: [SOURce<HW>]:ILS:GS:PRESet
driver.source.ils.gs.preset_with_opc()
```

Sets the parameters of the ILS glide slope component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_mode**(*mode: AvionicIlsGsMode*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:MODE
driver.source.ils.gs.set_mode(mode = enums.AvionicIlsGsMode.LLOBe)
```

Sets the operating mode for the ILS glide slope modulation signal.

**param mode**
NORM| ULOBe| LLOBe NORM ILS glide slope modulation is active. ULOBe Amplitude modulation of the output signal with the upper lobe (90Hz) signal component of the ILS glide slope signal is active. LLOBe Amplitude modulation of the output signal with the lower lobe (150Hz) signal component of the ILS glide slope signal is active.

**set_phase**(*phase: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:PHASe
driver.source.ils.gs.set_phase(phase = 1.0)
```

Sets the phase between the modulation signals of the upper and lower antenna lobe of the ILS glide slope signal. Zero crossing of the lower lobe (150Hz) signal serves as a reference. The angle refers to the period of the signal of the right antenna lobe.

**param phase**
float Range: -60 to 120

**set_sdm**(*sdm: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:SDM
driver.source.ils.gs.set_sdm(sdm = 1.0)
```

Sets the arithmetic sum of the modulation depths of the upper lobe (90 Hz) and lower lobe (150 Hz) for the ILS glide slope signal contents. The RMS modulation depth of the sum signal depends on the phase setting of both modulation tones.

**param sdm**
float Range: 0 to 100

**set_source**(*ils_gs_source: AvionicExtAm*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:SOURce
driver.source.ils.gs.set_source(ils_gs_source = enums.AvionicExtAm.EXT)
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**param ils_gs_source**
INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:ILS:GS:STATe
driver.source.ils.gs.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.11.1.1 Ddm

### SCPI Commands :

```
[SOURce<HW>]:ILS:[GS]:DDM:COUPling
[SOURce<HW>]:ILS:[GS]:DDM:CURRent
[SOURce<HW>]:ILS:[GS]:DDM:DIRection
[SOURce<HW>]:ILS:[GS]:DDM:LOGarithmic
[SOURce<HW>]:ILS:[GS]:DDM:PCT
[SOURce<HW>]:ILS:[GS]:DDM:POLarity
[SOURce<HW>]:ILS:[GS]:DDM:STEP
[SOURce<HW>]:ILS:[GS]:DDM:[DEPTh]
```

**class DdmCls**

Ddm commands group definition. 8 total commands, 0 Subgroups, 8 group commands

**get_coupling**() → AvionicIlsDdmCoup

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:COUPling
value: enums.AvionicIlsDdmCoup = driver.source.ils.gs.ddm.get_coupling()
```

Selects if the DDM value is fixed or is changed with a change of sum of modulation depths (SDM, see [:SOURce<hw>]:ILS[:GS|GSLope]:SDM) .

**return**
coupling: FIXed| SDM

**get_current**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:CURRent
value: float = driver.source.ils.gs.ddm.get_current()
```

Sets the DDM value alternatively as a current by means of the ILS indicating instrument. The instrument current is calculated according to: DDM Current uA = DDM Depth [%] x 857,125 uA A variation of the instrument current automatically leads to a variation of the DDM value and the DDM value in dB.

**return**
current: float Range: -8.57125E-4 to 8.57125E-4

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:[DEPTh]
value: float = driver.source.ils.gs.ddm.get_depth()
```

Sets the difference in depth of modulation between the signal of the upper/left lobe (90 Hz) and the lower/right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. The following is true: ILS:GS|GSL:DDM:DEPTh = (AM(90Hz) - AM(150Hz) )/100% A variation of the DDM value automatically leads to a variation of the DDM value in dB and the value of the instrument current.

**return**
depth: float Range: -0.8 to 0.8

**get_direction**() → UpDownDirection

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:DIRection
value: enums.UpDownDirection = driver.source.ils.gs.ddm.get_direction()
```

Sets the simulation mode for the ILS glide slope modulation signal. A change of the setting automatically changes the sign of the DDM value.

**return**
direction: UP| DOWN UP The 150-Hz modulation signal is predominant, the DDM value is negative (the airplane is too low, it must climb) . DOWN The 90-Hz modulation signal is predominant, the DDM value is positive (the airplane is too high, it must descend) .

**get_logarithmic**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:LOGarithmic
value: float = driver.source.ils.gs.ddm.get_logarithmic()
```

Sets the depth of modulation value for ILS glide slope modulation in dB. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**return**
logarithmic: float Range: -999.9 to 999.9

**get_pct**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:PCT
value: float = driver.source.ils.gs.ddm.get_pct()
```

Sets the difference in depth of modulation between the signal of the upper lobe (90 Hz) and the lower lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**return**
pct: float Range: -80.0 to 80.0

**get_polarity**() → AvionicIlsDdmPol

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:POLarity
value: enums.AvionicIlsDdmPol = driver.source.ils.gs.ddm.get_polarity()
```

Sets the polarity for DDM calculation (see [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh]) . The DDM depth calculation depends on the selected polarity:

INTRO_CMD_HELP: The effect depends on the selected mode:

- Polarity 90 Hz - 150 Hz (default setting) : DDM = [ AM(90 Hz) - AM (150 Hz) ] / 100%
- Polarity 150 Hz - 90 Hz: DDM = [ AM(150 Hz) - AM (90 Hz) ] / 100%

**return**
polarity: P90_150| P150_90

**get_step**() → AvionicDdmStep

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:STEP
value: enums.AvionicDdmStep = driver.source.ils.gs.ddm.get_step()
```

Sets the variation of the difference in depth of modulation via the rotary knob.

**return**
ddm_step: DECimal| PREDefined

**set_coupling**(*coupling: AvionicIlsDdmCoup*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:COUPling
driver.source.ils.gs.ddm.set_coupling(coupling = enums.AvionicIlsDdmCoup.FIXed)
```

Selects if the DDM value is fixed or is changed with a change of sum of modulation depths (SDM, see [:SOURce<hw>]:ILS[:GS|GSLope]:SDM) .

**param coupling**
FIXed| SDM

**set_current**(*current: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:CURRent
driver.source.ils.gs.ddm.set_current(current = 1.0)
```

Sets the DDM value alternatively as a current by means of the ILS indicating instrument. The instrument current is calculated according to: DDM Current uA = DDM Depth [%] x 857,125 uA A variation of the instrument current automatically leads to a variation of the DDM value and the DDM value in dB.

**param current**
float Range: -8.57125E-4 to 8.57125E-4

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:[DEPTh]
driver.source.ils.gs.ddm.set_depth(depth = 1.0)
```

Sets the difference in depth of modulation between the signal of the upper/left lobe (90 Hz) and the lower/right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. The following is true: ILS:GS|GSL:DDM:DEPTh = (AM(90Hz) - AM(150Hz) )/100% A variation of the DDM value automatically leads to a variation of the DDM value in dB and the value of the instrument current.

**param depth**
float Range: -0.8 to 0.8

**set_direction**(*direction: UpDownDirection*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:DIRection
driver.source.ils.gs.ddm.set_direction(direction = enums.UpDownDirection.DOWN)
```

Sets the simulation mode for the ILS glide slope modulation signal. A change of the setting automatically changes the sign of the DDM value.

**param direction**
UP| DOWN UP The 150-Hz modulation signal is predominant, the DDM value is negative (the airplane is too low, it must climb) . DOWN The 90-Hz modulation signal is predominant, the DDM value is positive (the airplane is too high, it must descend) .

**set_logarithmic**(*logarithmic: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:LOGarithmic
driver.source.ils.gs.ddm.set_logarithmic(logarithmic = 1.0)
```

Sets the depth of modulation value for ILS glide slope modulation in dB. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**param logarithmic**
float Range: -999.9 to 999.9

**set_pct**(*pct: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:PCT
driver.source.ils.gs.ddm.set_pct(pct = 1.0)
```

Sets the difference in depth of modulation between the signal of the upper lobe (90 Hz) and the lower lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**param pct**
float Range: -80.0 to 80.0

**set_polarity**(*polarity: AvionicIlsDdmPol*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:POLarity
driver.source.ils.gs.ddm.set_polarity(polarity = enums.AvionicIlsDdmPol.P150_90)
```

Sets the polarity for DDM calculation (see [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh]) . The DDM depth calculation depends on the selected polarity:

INTRO_CMD_HELP: The effect depends on the selected mode:

- Polarity 90 Hz - 150 Hz (default setting) : DDM = [ AM(90 Hz) - AM (150 Hz) ] / 100%
- Polarity 150 Hz - 90 Hz: DDM = [ AM(150 Hz) - AM (90 Hz) ] / 100%

**param polarity**
P90_150| P150_90

**set_step**(*ddm_step: AvionicDdmStep*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:DDM:STEP
driver.source.ils.gs.ddm.set_step(ddm_step = enums.AvionicDdmStep.DECimal)
```

Sets the variation of the difference in depth of modulation via the rotary knob.

**param ddm_step**
DECimal| PREDefined

#### 6.18.11.1.2 Frequency

##### SCPI Commands :

```
[SOURce<HW>]:ILS:[GS]:FREQuency:MODE
[SOURce<HW>]:ILS:[GS]:FREQuency:STEP
[SOURce<HW>]:ILS:[GS]:FREQuency
```

**class FrequencyCls**

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

**get_mode**() → AvionicCarrFreqMode

```
# SCPI: [SOURce<HW>]:ILS:[GS]:FREQuency:MODE
value: enums.AvionicCarrFreqMode = driver.source.ils.gs.frequency.get_mode()
```

Sets the mode for the carrier frequency of the signal.

**return**
mode: USER| ICAO DECimal Activates user-defined variation of the carrier frequency. ICAO Activates variation in predefined steps according to standard ILS transmitting frequencies (see Table 'ILS ICAO channels and frequencies (MHz) ') .

**get_step**() → AvionicKnobStep

```
# SCPI: [SOURce<HW>]:ILS:[GS]:FREQuency:STEP
value: enums.AvionicKnobStep = driver.source.ils.gs.frequency.get_step()
```

No command help available

**return**
step: No help available

**get_value**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:FREQuency
value: float = driver.source.ils.gs.frequency.get_value()
```

Sets the carrier frequency of the signal.

**return**
carrier_freq: float Range: 100E3 to 6E9

**set_mode**(*mode: AvionicCarrFreqMode*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:FREQuency:MODE
driver.source.ils.gs.frequency.set_mode(mode = enums.AvionicCarrFreqMode.
↪DECimal)
```

Sets the mode for the carrier frequency of the signal.

**param mode**
USER| ICAO DECimal Activates user-defined variation of the carrier frequency. ICAO Activates variation in predefined steps according to standard ILS transmitting frequencies (see Table 'ILS ICAO channels and frequencies (MHz) ') .

**set_step**(*step: AvionicKnobStep*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:FREQuency:STEP
driver.source.ils.gs.frequency.set_step(step = enums.AvionicKnobStep.DECimal)
```

No command help available

**param step**
No help available

**set_value**(*carrier_freq: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:FREQuency
driver.source.ils.gs.frequency.set_value(carrier_freq = 1.0)
```

Sets the carrier frequency of the signal.

**param carrier_freq**
float Range: 100E3 to 6E9

#### 6.18.11.1.3 Icao

#### SCPI Command :

```
[SOURce<HW>]:ILS:[GS]:ICAO:CHANnel
```

**class IcaoCls**

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

**get_channel**() → AvionicIlsIcaoChan

```
# SCPI: [SOURce<HW>]:ILS:[GS]:ICAO:CHANnel
value: enums.AvionicIlsIcaoChan = driver.source.ils.gs.icao.get_channel()
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The ICAO channel settings for ILS glide slope/localizer components are coupled. For an overview of the ILS ICAO channel frequencies, see Table 'ILS ICAO channels and frequencies (MHz) '.

**return**
channel: CH18X| CH18Y| CH20X| CH20Y| CH22X| CH22Y| CH24X| CH24Y| CH26X| CH26Y| CH28X| CH28Y| CH30X| CH30Y| CH32X| CH32Y| CH34X| CH34Y| CH36X| CH36Y| CH38X| CH38Y| CH40X| CH40Y| CH42X| CH42Y| CH44X| CH44Y| CH46X| CH46Y| CH48X| CH48Y| CH50X| CH50Y| CH52X| CH52Y| CH54X| CH54Y| CH56X| CH56Y

**set_channel**(*channel: AvionicIlsIcaoChan*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:ICAO:CHANnel
driver.source.ils.gs.icao.set_channel(channel = enums.AvionicIlsIcaoChan.CH18X)
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The ICAO channel settings for ILS glide slope/localizer components are coupled. For an overview of the ILS ICAO channel frequencies, see Table 'ILS ICAO channels and frequencies (MHz) '.

**param channel**
CH18X| CH18Y| CH20X| CH20Y| CH22X| CH22Y| CH24X| CH24Y| CH26X| CH26Y| CH28X| CH28Y| CH30X| CH30Y| CH32X| CH32Y| CH34X| CH34Y| CH36X| CH36Y| CH38X| CH38Y| CH40X| CH40Y| CH42X| CH42Y| CH44X| CH44Y| CH46X| CH46Y| CH48X| CH48Y| CH50X| CH50Y| CH52X| CH52Y| CH54X| CH54Y| CH56X| CH56Y

#### 6.18.11.1.4 Llobe

**SCPI Command :**

```
[SOURce<HW>]:ILS:[GS]:LLOBe:[FREQuency]
```

**class LlobeCls**

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

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:LLOBe:[FREQuency]
value: float = driver.source.ils.gs.llobe.get_frequency()
```

Sets the modulation frequency of the antenna lobe arranged at the bottom viewed from the air plane for the ILS glide slope modulation signal.

**return**
frequency: float Range: 100 to 200

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:LLOBe:[FREQuency]
driver.source.ils.gs.llobe.set_frequency(frequency = 1.0)
```

Sets the modulation frequency of the antenna lobe arranged at the bottom viewed from the air plane for the ILS glide slope modulation signal.

**param frequency**
float Range: 100 to 200

#### 6.18.11.1.5 Ulobe

**SCPI Command :**

```
[SOURce<HW>]:ILS:[GS]:ULOBe:[FREQuency]
```

**class UlobeCls**

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

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GS]:ULOBe:[FREQuency]
value: float = driver.source.ils.gs.ulobe.get_frequency()
```

No command help available

**return**
frequency: float Range: 60 to 120

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GS]:ULOBe:[FREQuency]
driver.source.ils.gs.ulobe.set_frequency(frequency = 1.0)
```

No command help available

**param frequency**
float Range: 60 to 120

### 6.18.11.2 Gslope

**SCPI Commands :**

```
[SOURce<HW>]:ILS:GSLope:PRESet
[SOURce<HW>]:ILS:GSLope:STATe
[SOURce<HW>]:ILS:[GSLope]:MODE
[SOURce<HW>]:ILS:[GSLope]:PHASe
[SOURce<HW>]:ILS:[GSLope]:SDM
[SOURce<HW>]:ILS:[GSLope]:SOURce
```

**class GslopeCls**

Gslope commands group definition. 20 total commands, 5 Subgroups, 6 group commands

**get_mode**() → AvionicIlsGsMode

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:MODE
value: enums.AvionicIlsGsMode = driver.source.ils.gslope.get_mode()
```

Sets the operating mode for the ILS glide slope modulation signal.

**return**
mode: NORM| ULOBe| LLOBe NORM ILS glide slope modulation is active. ULOBe Amplitude modulation of the output signal with the upper lobe (90Hz) signal component of the ILS glide slope signal is active. LLOBe Amplitude modulation of the output

signal with the lower lobe (150Hz) signal component of the ILS glide slope signal is active.

**get_phase**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:PHASe
value: float = driver.source.ils.gslope.get_phase()
```

Sets the phase between the modulation signals of the upper and lower antenna lobe of the ILS glide slope signal. Zero crossing of the lower lobe (150Hz) signal serves as a reference. The angle refers to the period of the signal of the right antenna lobe.

**return**
phase: float Range: -60 to 120

**get_sdm**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:SDM
value: float = driver.source.ils.gslope.get_sdm()
```

Sets the arithmetic sum of the modulation depths of the upper lobe (90 Hz) and lower lobe (150 Hz) for the ILS glide slope signal contents. The RMS modulation depth of the sum signal depends on the phase setting of both modulation tones.

**return**
sdm: float Range: 0 to 100

**get_source**() → AvionicExtAm

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:SOURce
value: enums.AvionicExtAm = driver.source.ils.gslope.get_source()
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**return**
ils_gs_source: INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:ILS:GSLope:STATe
value: bool = driver.source.ils.gslope.get_state()
```

No command help available

**return**
state: No help available

**preset**() → None

```
# SCPI: [SOURce<HW>]:ILS:GSLope:PRESet
driver.source.ils.gslope.preset()
```

Sets the parameters of the ILS glide slope component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

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

```
# SCPI: [SOURce<HW>]:ILS:GSLope:PRESet
driver.source.ils.gslope.preset_with_opc()
```

Sets the parameters of the ILS glide slope component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_mode**(*mode: AvionicIlsGsMode*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:MODE
driver.source.ils.gslope.set_mode(mode = enums.AvionicIlsGsMode.LLOBe)
```

Sets the operating mode for the ILS glide slope modulation signal.

**param mode**
NORM| ULOBe| LLOBe NORM ILS glide slope modulation is active. ULOBe Amplitude modulation of the output signal with the upper lobe (90Hz) signal component of the ILS glide slope signal is active. LLOBe Amplitude modulation of the output signal with the lower lobe (150Hz) signal component of the ILS glide slope signal is active.

**set_phase**(*phase: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:PHASe
driver.source.ils.gslope.set_phase(phase = 1.0)
```

Sets the phase between the modulation signals of the upper and lower antenna lobe of the ILS glide slope signal. Zero crossing of the lower lobe (150Hz) signal serves as a reference. The angle refers to the period of the signal of the right antenna lobe.

**param phase**
float Range: -60 to 120

**set_sdm**(*sdm: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:SDM
driver.source.ils.gslope.set_sdm(sdm = 1.0)
```

Sets the arithmetic sum of the modulation depths of the upper lobe (90 Hz) and lower lobe (150 Hz) for the ILS glide slope signal contents. The RMS modulation depth of the sum signal depends on the phase setting of both modulation tones.

**param sdm**
float Range: 0 to 100

**set_source**(*ils_gs_source: AvionicExtAm*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:SOURce
driver.source.ils.gslope.set_source(ils_gs_source = enums.AvionicExtAm.EXT)
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**param ils_gs_source**
INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:ILS:GSLope:STATe
driver.source.ils.gslope.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.11.2.1 Ddm

### SCPI Commands :

```
[SOURce<HW>]:ILS:[GSLope]:DDM:COUPling
[SOURce<HW>]:ILS:[GSLope]:DDM:CURRent
[SOURce<HW>]:ILS:[GSLope]:DDM:DIRection
[SOURce<HW>]:ILS:[GSLope]:DDM:LOGarithmic
[SOURce<HW>]:ILS:[GSLope]:DDM:PCT
[SOURce<HW>]:ILS:[GSLope]:DDM:POLarity
[SOURce<HW>]:ILS:[GSLope]:DDM:STEP
[SOURce<HW>]:ILS:[GSLope]:DDM:[DEPTh]
```

**class DdmCls**

Ddm commands group definition. 8 total commands, 0 Subgroups, 8 group commands

**get_coupling**() → AvionicIlsDdmCoup

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:COUPling
value: enums.AvionicIlsDdmCoup = driver.source.ils.gslope.ddm.get_coupling()
```

Selects if the DDM value is fixed or is changed with a change of sum of modulation depths (SDM, see [:SOURce<hw>]:ILS[:GS|GSLope]:SDM) .

**return**
coupling: FIXed| SDM

**get_current**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:CURRent
value: float = driver.source.ils.gslope.ddm.get_current()
```

Sets the DDM value alternatively as a current by means of the ILS indicating instrument. The instrument current is calculated according to: DDM Current uA = DDM Depth [%] x 857,125 uA A variation of the instrument current automatically leads to a variation of the DDM value and the DDM value in dB.

**return**
current: float Range: -8.57125E-4 to 8.57125E-4

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:[DEPTh]
value: float = driver.source.ils.gslope.ddm.get_depth()
```

Sets the difference in depth of modulation between the signal of the upper/left lobe (90 Hz) and the lower/right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. The following is true: ILS:GS|GSL:DDM:DEPTh = (AM(90Hz) - AM(150Hz) )/100% A variation of the DDM value automatically leads to a variation of the DDM value in dB and the value of the instrument current.

**return**
depth: float Range: -0.8 to 0.8

**get_direction**() → UpDownDirection

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:DIRection
value: enums.UpDownDirection = driver.source.ils.gslope.ddm.get_direction()
```

Sets the simulation mode for the ILS glide slope modulation signal. A change of the setting automatically changes the sign of the DDM value.

**return**
direction: UP| DOWN UP The 150-Hz modulation signal is predominant, the DDM value is negative (the airplane is too low, it must climb) . DOWN The 90-Hz modulation signal is predominant, the DDM value is positive (the airplane is too high, it must descend) .

**get_logarithmic**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:LOGarithmic
value: float = driver.source.ils.gslope.ddm.get_logarithmic()
```

Sets the depth of modulation value for ILS glide slope modulation in dB. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**return**
logarithmic: float Range: -999.9 to 999.9

**get_pct**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:PCT
value: float = driver.source.ils.gslope.ddm.get_pct()
```

Sets the difference in depth of modulation between the signal of the upper lobe (90 Hz) and the lower lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**return**
pct: float Range: -80.0 to 80.0

**get_polarity**() → AvionicIlsDdmPol

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:POLarity
value: enums.AvionicIlsDdmPol = driver.source.ils.gslope.ddm.get_polarity()
```

Sets the polarity for DDM calculation (see [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh]) . The DDM depth calculation depends on the selected polarity:

INTRO_CMD_HELP: The effect depends on the selected mode:

- Polarity 90 Hz - 150 Hz (default setting) : DDM = [ AM(90 Hz) - AM (150 Hz) ] / 100%
- Polarity 150 Hz - 90 Hz: DDM = [ AM(150 Hz) - AM (90 Hz) ] / 100%

**return**
polarity: P90_150| P150_90

**get_step**() → AvionicDdmStep

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:STEP
value: enums.AvionicDdmStep = driver.source.ils.gslope.ddm.get_step()
```

Sets the variation of the difference in depth of modulation via the rotary knob.

**return**
ddm_step: DECimal| PREDefined

**set_coupling**(*coupling: AvionicIlsDdmCoup*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:COUPling
driver.source.ils.gslope.ddm.set_coupling(coupling = enums.AvionicIlsDdmCoup.
↪FIXed)
```

Selects if the DDM value is fixed or is changed with a change of sum of modulation depths (SDM, see [:SOURce<hw>]:ILS[:GS|GSLope]:SDM) .

**param coupling**
FIXed| SDM

**set_current**(*current: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:CURRent
driver.source.ils.gslope.ddm.set_current(current = 1.0)
```

Sets the DDM value alternatively as a current by means of the ILS indicating instrument. The instrument current is calculated according to: DDM Current uA = DDM Depth [%] x 857,125 uA A variation of the instrument current automatically leads to a variation of the DDM value and the DDM value in dB.

**param current**
float Range: -8.57125E-4 to 8.57125E-4

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:[DEPTh]
driver.source.ils.gslope.ddm.set_depth(depth = 1.0)
```

Sets the difference in depth of modulation between the signal of the upper/left lobe (90 Hz) and the lower/right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. The following is true: ILS:GS|GSL:DDM:DEPTh = (AM(90Hz) - AM(150Hz) )/100%

A variation of the DDM value automatically leads to a variation of the DDM value in dB and the value of the instrument current.

**param depth**
float Range: -0.8 to 0.8

**set_direction**(*direction: UpDownDirection*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:DIRection
driver.source.ils.gslope.ddm.set_direction(direction = enums.UpDownDirection.
↪DOWN)
```

Sets the simulation mode for the ILS glide slope modulation signal. A change of the setting automatically changes the sign of the DDM value.

**param direction**
UP| DOWN UP The 150-Hz modulation signal is predominant, the DDM value is negative (the airplane is too low, it must climb) . DOWN The 90-Hz modulation signal is predominant, the DDM value is positive (the airplane is too high, it must descend) .

**set_logarithmic**(*logarithmic: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:LOGarithmic
driver.source.ils.gslope.ddm.set_logarithmic(logarithmic = 1.0)
```

Sets the depth of modulation value for ILS glide slope modulation in dB. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**param logarithmic**
float Range: -999.9 to 999.9

**set_pct**(*pct: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:PCT
driver.source.ils.gslope.ddm.set_pct(pct = 1.0)
```

Sets the difference in depth of modulation between the signal of the upper lobe (90 Hz) and the lower lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. See also [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh].

**param pct**
float Range: -80.0 to 80.0

**set_polarity**(*polarity: AvionicIlsDdmPol*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:POLarity
driver.source.ils.gslope.ddm.set_polarity(polarity = enums.AvionicIlsDdmPol.
↪P150_90)
```

Sets the polarity for DDM calculation (see [:SOURce<hw>]:ILS[:GS|GSLope]:DDM[:DEPTh]) . The DDM depth calculation depends on the selected polarity:

INTRO_CMD_HELP: The effect depends on the selected mode:

- Polarity 90 Hz - 150 Hz (default setting) : DDM = [ AM(90 Hz) - AM (150 Hz) ] / 100%
- Polarity 150 Hz - 90 Hz: DDM = [ AM(150 Hz) - AM (90 Hz) ] / 100%

**param polarity**
P90_150| P150_90

**set_step**(*ddm_step: AvionicDdmStep*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:DDM:STEP
driver.source.ils.gslope.ddm.set_step(ddm_step = enums.AvionicDdmStep.DECimal)
```

Sets the variation of the difference in depth of modulation via the rotary knob.

**param ddm_step**
DECimal| PREDefined

#### 6.18.11.2.2 Frequency

##### SCPI Commands :

```
[SOURce<HW>]:ILS:[GSLope]:FREQuency:MODE
[SOURce<HW>]:ILS:[GSLope]:FREQuency:STEP
[SOURce<HW>]:ILS:[GSLope]:FREQuency
```

**class FrequencyCls**

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

**get_mode**() → AvionicCarrFreqMode

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:FREQuency:MODE
value: enums.AvionicCarrFreqMode = driver.source.ils.gslope.frequency.get_mode()
```

Sets the mode for the carrier frequency of the signal.

**return**
mode: USER| ICAO DECimal Activates user-defined variation of the carrier frequency. ICAO Activates variation in predefined steps according to standard ILS transmitting frequencies (see Table 'ILS ICAO channels and frequencies (MHz) ') .

**get_step**() → AvionicKnobStep

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:FREQuency:STEP
value: enums.AvionicKnobStep = driver.source.ils.gslope.frequency.get_step()
```

No command help available

**return**
step: No help available

**get_value**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:FREQuency
value: float = driver.source.ils.gslope.frequency.get_value()
```

Sets the carrier frequency of the signal.

**return**
carrier_freq: float Range: 100E3 to 6E9

**set_mode**(*mode: AvionicCarrFreqMode*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:FREQuency:MODE
driver.source.ils.gslope.frequency.set_mode(mode = enums.AvionicCarrFreqMode.
↪DECimal)
```

Sets the mode for the carrier frequency of the signal.

**param mode**
USER| ICAO DECimal Activates user-defined variation of the carrier frequency. ICAO Activates variation in predefined steps according to standard ILS transmitting frequencies (see Table 'ILS ICAO channels and frequencies (MHz) ') .

**set_step**(*step: AvionicKnobStep*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:FREQuency:STEP
driver.source.ils.gslope.frequency.set_step(step = enums.AvionicKnobStep.
↪DECimal)
```

No command help available

**param step**
No help available

**set_value**(*carrier_freq: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:FREQuency
driver.source.ils.gslope.frequency.set_value(carrier_freq = 1.0)
```

Sets the carrier frequency of the signal.

**param carrier_freq**
float Range: 100E3 to 6E9

#### 6.18.11.2.3 Icao

#### SCPI Command :

```
[SOURce<HW>]:ILS:[GSLope]:ICAO:CHANnel
```

**class IcaoCls**

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

**get_channel**() → AvionicIlsIcaoChan

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:ICAO:CHANnel
value: enums.AvionicIlsIcaoChan = driver.source.ils.gslope.icao.get_channel()
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The ICAO channel settings for ILS glide slope/localizer components are coupled. For an overview of the ILS ICAO channel frequencies, see Table 'ILS ICAO channels and frequencies (MHz) '.

**return**
channel: CH18X| CH18Y| CH20X| CH20Y| CH22X| CH22Y| CH24X| CH24Y| CH26X| CH26Y| CH28X| CH28Y| CH30X| CH30Y| CH32X| CH32Y| CH34X|

CH34Y| CH36X| CH36Y| CH38X| CH38Y| CH40X| CH40Y| CH42X| CH42Y| CH44X| CH44Y| CH46X| CH46Y| CH48X| CH48Y| CH50X| CH50Y| CH52X| CH52Y| CH54X| CH54Y| CH56X| CH56Y

**set_channel**(*channel: AvionicIlsIcaoChan*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:ICAO:CHANnel
driver.source.ils.gslope.icao.set_channel(channel = enums.AvionicIlsIcaoChan.
↪CH18X)
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The ICAO channel settings for ILS glide slope/localizer components are coupled. For an overview of the ILS ICAO channel frequencies, see Table 'ILS ICAO channels and frequencies (MHz) '.

**param channel**
CH18X| CH18Y| CH20X| CH20Y| CH22X| CH22Y| CH24X| CH24Y| CH26X| CH26Y| CH28X| CH28Y| CH30X| CH30Y| CH32X| CH32Y| CH34X| CH34Y| CH36X| CH36Y| CH38X| CH38Y| CH40X| CH40Y| CH42X| CH42Y| CH44X| CH44Y| CH46X| CH46Y| CH48X| CH48Y| CH50X| CH50Y| CH52X| CH52Y| CH54X| CH54Y| CH56X| CH56Y

#### 6.18.11.2.4 Llobe

**SCPI Command :**

```
[SOURce<HW>]:ILS:[GSLope]:LLOBe:[FREQuency]
```

**class LlobeCls**

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

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:LLOBe:[FREQuency]
value: float = driver.source.ils.gslope.llobe.get_frequency()
```

Sets the modulation frequency of the antenna lobe arranged at the bottom viewed from the air plane for the ILS glide slope modulation signal.

**return**
frequency: float Range: 100 to 200

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:LLOBe:[FREQuency]
driver.source.ils.gslope.llobe.set_frequency(frequency = 1.0)
```

Sets the modulation frequency of the antenna lobe arranged at the bottom viewed from the air plane for the ILS glide slope modulation signal.

**param frequency**
float Range: 100 to 200

#### 6.18.11.2.5 Ulobe

**SCPI Command :**

```
[SOURce<HW>]:ILS:[GSLope]:ULOBe:[FREQuency]
```

**class UlobeCls**

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

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:ULOBe:[FREQuency]
value: float = driver.source.ils.gslope.ulobe.get_frequency()
```

No command help available

**return**
frequency: float Range: 60 to 120

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:[GSLope]:ULOBe:[FREQuency]
driver.source.ils.gslope.ulobe.set_frequency(frequency = 1.0)
```

No command help available

**param frequency**
float Range: 60 to 120

### 6.18.11.3 Localizer

**SCPI Commands :**

```
[SOURce<HW>]:ILS:LOCalizer:MODE
[SOURce<HW>]:ILS:LOCalizer:PHASe
[SOURce<HW>]:ILS:LOCalizer:PRESet
[SOURce<HW>]:ILS:LOCalizer:SDM
[SOURce<HW>]:ILS:LOCalizer:SOURce
[SOURce<HW>]:ILS:LOCalizer:STATe
```

**class LocalizerCls**

Localizer commands group definition. 32 total commands, 6 Subgroups, 6 group commands

**get_mode**() → AvionicIlsLocMode

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:MODE
value: enums.AvionicIlsLocMode = driver.source.ils.localizer.get_mode()
```

Sets the operating mode for the ILS localizer modulation signal.

**return**
mode: NORM| LLOBe| RLOBe NORM ILS localizer modulation is active. LLOBe Amplitude modulation of the output signal with the left lobe (90Hz) signal component of the ILS localizer signal is active. RLOBe Amplitude modulation of the output signal with the right lobe (150Hz) signal component of the ILS localizer signal is active.

**get_phase**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:PHASe
value: float = driver.source.ils.localizer.get_phase()
```

Sets the phase between the modulation signals of the left and right antenna lobe of the ILS localizer signal. The zero crossing of the right lobe (150Hz) signal serves as a reference. The angle refers to the period of the signal of the right antenna lobe.

**return**
phase: float Range: -60 to 120

**get_sdm**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:SDM
value: float = driver.source.ils.localizer.get_sdm()
```

Sets the arithmetic sum of the modulation depths of the left lobe (90 Hz) and right lobe (150 Hz) for the ILS localizer signal contents. The RMS modulation depth of the sum signal depends on the phase setting of both modulation tones.

**return**
sdm: float Range: 0 to 100

**get_source**() → AvionicExtAm

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:SOURce
value: enums.AvionicExtAm = driver.source.ils.localizer.get_source()
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**return**
ils_loc_source: INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:STATe
value: bool = driver.source.ils.localizer.get_state()
```

No command help available

**return**
state: No help available

**preset**() → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:PRESet
driver.source.ils.localizer.preset()
```

Sets the parameters of the ILS localizer component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

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

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:PRESet
driver.source.ils.localizer.preset_with_opc()
```

Sets the parameters of the ILS localizer component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_mode**(*mode: AvionicIlsLocMode*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:MODE
driver.source.ils.localizer.set_mode(mode = enums.AvionicIlsLocMode.LLOBe)
```

Sets the operating mode for the ILS localizer modulation signal.

**param mode**
NORM| LLOBe| RLOBe NORM ILS localizer modulation is active. LLOBe Amplitude modulation of the output signal with the left lobe (90Hz) signal component of the ILS localizer signal is active. RLOBe Amplitude modulation of the output signal with the right lobe (150Hz) signal component of the ILS localizer signal is active.

**set_phase**(*phase: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:PHASe
driver.source.ils.localizer.set_phase(phase = 1.0)
```

Sets the phase between the modulation signals of the left and right antenna lobe of the ILS localizer signal. The zero crossing of the right lobe (150Hz) signal serves as a reference. The angle refers to the period of the signal of the right antenna lobe.

**param phase**
float Range: -60 to 120

**set_sdm**(*sdm: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:SDM
driver.source.ils.localizer.set_sdm(sdm = 1.0)
```

Sets the arithmetic sum of the modulation depths of the left lobe (90 Hz) and right lobe (150 Hz) for the ILS localizer signal contents. The RMS modulation depth of the sum signal depends on the phase setting of both modulation tones.

**param sdm**
float Range: 0 to 100

**set_source**(*ils_loc_source: AvionicExtAm*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:SOURce
driver.source.ils.localizer.set_source(ils_loc_source = enums.AvionicExtAm.EXT)
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**param ils_loc_source**
INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:STATe
driver.source.ils.localizer.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.11.3.1 Comid

### SCPI Commands :

```
[SOURce<HW>]:ILS:LOCalizer:COMid:DASH
[SOURce<HW>]:ILS:LOCalizer:COMid:DEPTh
[SOURce<HW>]:ILS:LOCalizer:COMid:DOT
[SOURce<HW>]:ILS:LOCalizer:COMid:FREQuency
[SOURce<HW>]:ILS:LOCalizer:COMid:LETTer
[SOURce<HW>]:ILS:LOCalizer:COMid:PERiod
[SOURce<HW>]:ILS:LOCalizer:COMid:REPeat
[SOURce<HW>]:ILS:LOCalizer:COMid:SYMBol
[SOURce<HW>]:ILS:LOCalizer:COMid:TSCHema
[SOURce<HW>]:ILS:LOCalizer:COMid:[STATe]
```

**class ComidCls**

Comid commands group definition. 12 total commands, 1 Subgroups, 10 group commands

**get_dash**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:DASH
value: float = driver.source.ils.localizer.comid.get_dash()
```

Sets the length of a Morse code dash.

**return**
dash: float Range: 0.05 to 1

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:DEPTh
value: float = driver.source.ils.localizer.comid.get_depth()
```

Sets the AM modulation depth of the COM/ID signal.

**return**
depth: float Range: 0 to 100

**get_dot**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:DOT
value: float = driver.source.ils.localizer.comid.get_dot()
```

Sets the length of a Morse code dot.

**return**
dot: float Range: 0.05 to 1

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:FREQuency
value: float = driver.source.ils.localizer.comid.get_frequency()
```

Sets the frequency of the COM/ID signal.

**return**
frequency: float Range: 0.1 to 20E3

**get_letter**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:LETTer
value: float = driver.source.ils.localizer.comid.get_letter()
```

Sets the length of a Morse code letter space.

**return**
letter: float Range: 0.05 to 1

**get_period**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:PERiod
value: float = driver.source.ils.localizer.comid.get_period()
```

Sets the period of the COM/ID signal.

**return**
period: float Range: 0 to 120

**get_repeat**() → int

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:REPeat
value: int = driver.source.ils.localizer.comid.get_repeat()
```

No command help available

**return**
repeat: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:[STATe]
value: bool = driver.source.ils.localizer.comid.get_state()
```

Enables/disables the COM/ID signal.

**return**
state: 1| ON| 0| OFF

**get_symbol**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:SYMBol
value: float = driver.source.ils.localizer.comid.get_symbol()
```

Sets the length of the Morse code symbol space.

**return**
symbol: float Range: 0.05 to 1

**get_tschema**() → AvionicComIdTimeSchem

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:TSCHema
value: enums.AvionicComIdTimeSchem = driver.source.ils.localizer.comid.get_
↪tschema()
```

Sets the time schema of the Morse code for the COM/ID signal.

**return**
tschema: STD| USER STD Activates the standard time schema of the Morse code. The set dot length determines the dash length, which is 3 times the dot length. USER Activates the user-defined time schema of the Morse code. Dot and dash length, as well as symbol and letter space can be set separately.

**set_dash**(*dash: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:DASH
driver.source.ils.localizer.comid.set_dash(dash = 1.0)
```

Sets the length of a Morse code dash.

**param dash**
float Range: 0.05 to 1

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:DEPTh
driver.source.ils.localizer.comid.set_depth(depth = 1.0)
```

Sets the AM modulation depth of the COM/ID signal.

**param depth**
float Range: 0 to 100

**set_dot**(*dot: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:DOT
driver.source.ils.localizer.comid.set_dot(dot = 1.0)
```

Sets the length of a Morse code dot.

**param dot**
float Range: 0.05 to 1

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:FREQuency
driver.source.ils.localizer.comid.set_frequency(frequency = 1.0)
```

Sets the frequency of the COM/ID signal.

**param frequency**
float Range: 0.1 to 20E3

**set_letter**(*letter: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:LETTer
driver.source.ils.localizer.comid.set_letter(letter = 1.0)
```

Sets the length of a Morse code letter space.

**param letter**
float Range: 0.05 to 1

**set_period**(*period: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:PERiod
driver.source.ils.localizer.comid.set_period(period = 1.0)
```

Sets the period of the COM/ID signal.

**param period**
float Range: 0 to 120

**set_repeat**(*repeat: int*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:REPeat
driver.source.ils.localizer.comid.set_repeat(repeat = 1)
```

No command help available

**param repeat**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:[STATe]
driver.source.ils.localizer.comid.set_state(state = False)
```

Enables/disables the COM/ID signal.

**param state**
1| ON| 0| OFF

**set_symbol**(*symbol: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:SYMBol
driver.source.ils.localizer.comid.set_symbol(symbol = 1.0)
```

Sets the length of the Morse code symbol space.

**param symbol**
float Range: 0.05 to 1

**set_tschema**(*tschema: AvionicComIdTimeSchem*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:TSCHema
driver.source.ils.localizer.comid.set_tschema(tschema = enums.
↪AvionicComIdTimeSchem.STD)
```

Sets the time schema of the Morse code for the COM/ID signal.

**param tschema**
STD| USER STD Activates the standard time schema of the Morse code. The set dot length determines the dash length, which is 3 times the dot length. USER Activates the user-defined time schema of the Morse code. Dot and dash length, as well as symbol and letter space can be set separately.

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.ils.localizer.comid.clone()
```

### Subgroups

#### 6.18.11.3.1.1 Code

### SCPI Commands :

```
[SOURce<HW>]:[ILS]:LOCalizer:COMid:CODE:STATe
[SOURce<HW>]:ILS:LOCalizer:COMid:CODE
```

**class CodeCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:[ILS]:LOCalizer:COMid:CODE:STATe
value: bool = driver.source.ils.localizer.comid.code.get_state()
```

No command help available

**return**
state: No help available

**get_value**() → str

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:CODE
value: str = driver.source.ils.localizer.comid.code.get_value()
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**return**
code: string

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:LOCalizer:COMid:CODE:STATe
driver.source.ils.localizer.comid.code.set_state(state = False)
```

No command help available

**param state**
No help available

**set_value**(*code: str*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:COMid:CODE
driver.source.ils.localizer.comid.code.set_value(code = 'abc')
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**param code**
string

#### 6.18.11.3.2 Ddm

**SCPI Commands :**

```
[SOURce<HW>]:ILS:LOCalizer:DDM:COUPling
[SOURce<HW>]:ILS:LOCalizer:DDM:CURRent
[SOURce<HW>]:ILS:LOCalizer:DDM:DIRection
[SOURce<HW>]:ILS:LOCalizer:DDM:LOGarithmic
[SOURce<HW>]:ILS:LOCalizer:DDM:PCT
[SOURce<HW>]:ILS:LOCalizer:DDM:POLarity
[SOURce<HW>]:ILS:LOCalizer:DDM:STEP
[SOURce<HW>]:ILS:LOCalizer:DDM:[DEPTh]
```

**class DdmCls**

Ddm commands group definition. 8 total commands, 0 Subgroups, 8 group commands

**get_coupling**() → AvionicIlsDdmCoup

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:COUPling
value: enums.AvionicIlsDdmCoup = driver.source.ils.localizer.ddm.get_coupling()
```

Selects if the DDM value is fixed or is changed with a change of sum of modulation depths (SDM, see [:SOURce<hw>]:ILS:LOCalizer:SDM) .

**return**
coupling: FIXed| SDM

**get_current**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:CURRent
value: float = driver.source.ils.localizer.ddm.get_current()
```

Sets the DDM value alternatively as a current by means of the ILS indicating instrument. The instrument current is calculated according to: DDM uA = DDM x 857,1 uA A variation of the instrument current automatically leads to a variation of the DDM value and the DDM value in dB.

**return**
current: float Range: -9.6775E-4 to 9.6775E-4

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:[DEPTh]
value: float = driver.source.ils.localizer.ddm.get_depth()
```

Sets the difference in depth of modulation between the signal of the upper/left lobe (90 Hz) and the lower/right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone.The following is true: ILS:LOC:DDM:DEPTh = (AM(90Hz) - AM(150Hz) )/100% A variation of the DDM value automatically leads to a variation of the DDM value in dB and the value of the instrument current.

**return**
depth: float Range: -0.4 to 0.4

**get_direction**() → LeftRightDirection

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:DIRection
value: enums.LeftRightDirection = driver.source.ils.localizer.ddm.get_
↪direction()
```

Sets the simulation mode for the ILS-LOC modulation signal. A change of the setting automatically changes the sign of the DDM value.

**return**
direction: LEFT| RIGHt LEFT The 150 Hz modulation signal is predominant, the DDM value is negative (the airplane is too far to the right, it must turn to the left) . RIGHT The 90 Hz modulation signal is predominant, the DDM value is positive (the airplane is too far to the left, it must turn to the right) .

**get_logarithmic**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:LOGarithmic
value: float = driver.source.ils.localizer.ddm.get_logarithmic()
```

Sets the modulation depth in dB for ILS localizer modulation. See also [:SOURce<hw>]:ILS:LOCalizer:DDM[:DEPTh].

**return**
logarithmic: float Range: -999.9 to 999.9

**get_pct**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:PCT
value: float = driver.source.ils.localizer.ddm.get_pct()
```

Sets the difference in depth of modulation between the signal of the left lobe (90 Hz) and the right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. See also [:SOURce<hw>]:ILS:LOCalizer:DDM[:DEPTh].

**return**
pct: float Range: -80.0 to 80.0

**get_polarity**() → AvionicIlsDdmPol

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:POLarity
value: enums.AvionicIlsDdmPol = driver.source.ils.localizer.ddm.get_polarity()
```

Sets the polarity for DDM calculation (see [:SOURce<hw>]:ILS:LOCalizer:DDM[:DEPTh]) . The DDM depth calculation depends on the selected polarity:

INTRO_CMD_HELP: The effect depends on the selected mode:

- Polarity 90 Hz - 150 Hz (default setting) : DDM = [ AM (90 Hz) - AM (150 Hz) ] / 100%
- Polarity 150 Hz - 90 Hz: DDM = [ AM (150 Hz) - AM (90 Hz) ] / 100%

**return**
polarity: P90_150| P150_90

**get_step**() → AvionicDdmStep

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:STEP
value: enums.AvionicDdmStep = driver.source.ils.localizer.ddm.get_step()
```

Sets the variation step of the DDM values.

**return**
ddm_step: DECimal| PREDefined

**set_coupling**(*coupling: AvionicIlsDdmCoup*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:COUPling
driver.source.ils.localizer.ddm.set_coupling(coupling = enums.AvionicIlsDdmCoup.
↪FIXed)
```

Selects if the DDM value is fixed or is changed with a change of sum of modulation depths (SDM, see [:SOURce<hw>]:ILS:LOCalizer:SDM) .

**param coupling**
FIXed| SDM

**set_current**(*current: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:CURRent
driver.source.ils.localizer.ddm.set_current(current = 1.0)
```

Sets the DDM value alternatively as a current by means of the ILS indicating instrument. The instrument current is calculated according to: DDM uA = DDM x 857,1 uA A variation of the instrument current automatically leads to a variation of the DDM value and the DDM value in dB.

**param current**
float Range: -9.6775E-4 to 9.6775E-4

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:[DEPTh]
driver.source.ils.localizer.ddm.set_depth(depth = 1.0)
```

Sets the difference in depth of modulation between the signal of the upper/left lobe (90 Hz) and the lower/right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone.The following is true: ILS:LOC:DDM:DEPTh = (AM(90Hz) - AM(150Hz) )/100%

A variation of the DDM value automatically leads to a variation of the DDM value in dB and the value of the instrument current.

**param depth**
float Range: -0.4 to 0.4

**set_direction**(*direction: LeftRightDirection*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:DIRection
driver.source.ils.localizer.ddm.set_direction(direction = enums.
↪LeftRightDirection.LEFT)
```

Sets the simulation mode for the ILS-LOC modulation signal. A change of the setting automatically changes the sign of the DDM value.

**param direction**
LEFT| RIGHt LEFT The 150 Hz modulation signal is predominant, the DDM value is negative (the airplane is too far to the right, it must turn to the left) . RIGHT The 90 Hz modulation signal is predominant, the DDM value is positive (the airplane is too far to the left, it must turn to the right) .

**set_logarithmic**(*logarithmic: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:LOGarithmic
driver.source.ils.localizer.ddm.set_logarithmic(logarithmic = 1.0)
```

Sets the modulation depth in dB for ILS localizer modulation. See also [:SOURce<hw>]:ILS:LOCalizer:DDM[:DEPTh].

**param logarithmic**
float Range: -999.9 to 999.9

**set_pct**(*pct: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:PCT
driver.source.ils.localizer.ddm.set_pct(pct = 1.0)
```

Sets the difference in depth of modulation between the signal of the left lobe (90 Hz) and the right lobe (150 Hz) . The maximum value equals the sum of the modulation depths of the 90 Hz and the 150 Hz tone. See also [:SOURce<hw>]:ILS:LOCalizer:DDM[:DEPTh].

**param pct**
float Range: -80.0 to 80.0

**set_polarity**(*polarity: AvionicIlsDdmPol*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:POLarity
driver.source.ils.localizer.ddm.set_polarity(polarity = enums.AvionicIlsDdmPol.
↪P150_90)
```

Sets the polarity for DDM calculation (see [:SOURce<hw>]:ILS:LOCalizer:DDM[:DEPTh]) . The DDM depth calculation depends on the selected polarity:

INTRO_CMD_HELP: The effect depends on the selected mode:

- Polarity 90 Hz - 150 Hz (default setting) : DDM = [ AM (90 Hz) - AM (150 Hz) ] / 100%
- Polarity 150 Hz - 90 Hz: DDM = [ AM (150 Hz) - AM (90 Hz) ] / 100%

**param polarity**
P90_150| P150_90

**set_step**(*ddm_step: AvionicDdmStep*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:DDM:STEP
driver.source.ils.localizer.ddm.set_step(ddm_step = enums.AvionicDdmStep.
↪DECimal)
```

Sets the variation step of the DDM values.

**param ddm_step**
DECimal| PREDefined

#### 6.18.11.3.3 Frequency

##### SCPI Commands :

```
[SOURce<HW>]:ILS:LOCalizer:FREQuency:MODE
[SOURce<HW>]:ILS:LOCalizer:FREQuency:STEP
[SOURce<HW>]:ILS:LOCalizer:FREQuency
```

**class FrequencyCls**

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

**get_mode**() → AvionicCarrFreqMode

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:FREQuency:MODE
value: enums.AvionicCarrFreqMode = driver.source.ils.localizer.frequency.get_
↪mode()
```

Sets the mode for the carrier frequency of the signal.

**return**
ils_loc_freq_mode: No help available

**get_step**() → AvionicCarrFreqMode

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:FREQuency:STEP
value: enums.AvionicCarrFreqMode = driver.source.ils.localizer.frequency.get_
↪step()
```

No command help available

**return**
ils_loc_freq_mode: No help available

**get_value**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:FREQuency
value: float = driver.source.ils.localizer.frequency.get_value()
```

Sets the carrier frequency of the signal.

**return**
carrier_freq: float Range: 100E3 to 6E9

**set_mode**(*ils_loc_freq_mode: AvionicCarrFreqMode*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:FREQuency:MODE
driver.source.ils.localizer.frequency.set_mode(ils_loc_freq_mode = enums.
↪AvionicCarrFreqMode.DECimal)
```

Sets the mode for the carrier frequency of the signal.

**param ils_loc_freq_mode**
DECimal| ICAO DECimal Activates user-defined variation of the carrier frequency. ICAO Activates variation in predefined steps according to standard ILS transmitting frequencies (see Table 'ILS ICAO channels and frequencies (MHz) ') .

**set_step**(*ils_loc_freq_mode: AvionicCarrFreqMode*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:FREQuency:STEP
driver.source.ils.localizer.frequency.set_step(ils_loc_freq_mode = enums.
↪AvionicCarrFreqMode.DECimal)
```

No command help available

**param ils_loc_freq_mode**
No help available

**set_value**(*carrier_freq: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:FREQuency
driver.source.ils.localizer.frequency.set_value(carrier_freq = 1.0)
```

Sets the carrier frequency of the signal.

**param carrier_freq**
float Range: 100E3 to 6E9

#### 6.18.11.3.4 Icao

**SCPI Command :**

```
[SOURce<HW>]:ILS:LOCalizer:ICAO:CHANnel
```

**class IcaoCls**

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

**get_channel**() → AvionicIlsIcaoChan

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:ICAO:CHANnel
value: enums.AvionicIlsIcaoChan = driver.source.ils.localizer.icao.get_channel()
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The ICAO channel settings for ILS glide slope/localizer components are coupled. For an overview of the ILS ICAO channel frequencies, see Table 'ILS ICAO channels and frequencies (MHz) '.

**return**
sel_icao_chan: CH18X| CH18Y| CH20X| CH20Y| CH22X| CH22Y| CH24X| CH24Y| CH26X| CH26Y| CH28X| CH28Y| CH30X| CH30Y| CH32X| CH32Y| CH34X| CH34Y| CH36X| CH36Y| CH38X| CH38Y| CH40X| CH40Y| CH42X| CH42Y| CH44X| CH44Y| CH46X| CH46Y| CH48X| CH48Y| CH50X| CH50Y| CH52X| CH52Y| CH54X| CH54Y| CH56X| CH56Y

**set_channel**(*sel_icao_chan: AvionicIlsIcaoChan*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:ICAO:CHANnel
driver.source.ils.localizer.icao.set_channel(sel_icao_chan = enums.
↪AvionicIlsIcaoChan.CH18X)
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The ICAO channel settings for ILS glide slope/localizer components are coupled. For an overview of the ILS ICAO channel frequencies, see Table 'ILS ICAO channels and frequencies (MHz) '.

**param sel_icao_chan**
CH18X| CH18Y| CH20X| CH20Y| CH22X| CH22Y| CH24X| CH24Y| CH26X| CH26Y| CH28X| CH28Y| CH30X| CH30Y| CH32X| CH32Y| CH34X| CH34Y| CH36X| CH36Y| CH38X| CH38Y| CH40X| CH40Y| CH42X| CH42Y| CH44X| CH44Y| CH46X| CH46Y| CH48X| CH48Y| CH50X| CH50Y| CH52X| CH52Y| CH54X| CH54Y| CH56X| CH56Y

#### 6.18.11.3.5 Llobe

**SCPI Command :**

```
[SOURce<HW>]:ILS:LOCalizer:LLOBe:[FREQuency]
```

**class LlobeCls**

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

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:LLOBe:[FREQuency]
value: float = driver.source.ils.localizer.llobe.get_frequency()
```

Sets the modulation frequency of the antenna lobe arranged at the bottom viewed from the air plane for the ILS localizer modulation signal.

**return**
frequency: float Range: 60 to 120

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:LLOBe:[FREQuency]
driver.source.ils.localizer.llobe.set_frequency(frequency = 1.0)
```

Sets the modulation frequency of the antenna lobe arranged at the bottom viewed from the air plane for the ILS localizer modulation signal.

**param frequency**
float Range: 60 to 120

#### 6.18.11.3.6 Rlobe

**SCPI Command :**

```
[SOURce<HW>]:ILS:LOCalizer:RLOBe:[FREQuency]
```

**class RlobeCls**

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

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:RLOBe:[FREQuency]
value: float = driver.source.ils.localizer.rlobe.get_frequency()
```

Sets the modulation frequency of the antenna lobe arranged at the right viewed from the air plane.

**return**
frequency: float Range: 100 to 200

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:ILS:LOCalizer:RLOBe:[FREQuency]
driver.source.ils.localizer.rlobe.set_frequency(frequency = 1.0)
```

Sets the modulation frequency of the antenna lobe arranged at the right viewed from the air plane.

**param frequency**
float Range: 100 to 200

### 6.18.11.4 Mbeacon

**SCPI Command :**

```
[SOURce<HW>]:[ILS]:MBEacon:PRESet
```

**class MbeaconCls**

Mbeacon commands group definition. 17 total commands, 3 Subgroups, 1 group commands

**preset**() → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:PRESet
driver.source.ils.mbeacon.preset()
```

Sets the parameters of the ILS marker beacons component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

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

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:PRESet
driver.source.ils.mbeacon.preset_with_opc()
```

Sets the parameters of the ILS marker beacons component to their default values (*RST values specified for the commands) . For other ILS preset commands, see [:SOURce<hw>]:ILS:PRESet.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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.source.ils.mbeacon.clone()
```

### Subgroups

#### 6.18.11.4.1 Comid

### SCPI Commands :

```
[SOURce<HW>]:[ILS]:MBEacon:COMid:DASH
[SOURce<HW>]:[ILS]:MBEacon:COMid:DEPTh
[SOURce<HW>]:[ILS]:MBEacon:COMid:DOT
[SOURce<HW>]:[ILS]:MBEacon:COMid:FREQuency
[SOURce<HW>]:[ILS]:MBEacon:COMid:LETTer
[SOURce<HW>]:[ILS]:MBEacon:COMid:PERiod
[SOURce<HW>]:[ILS]:MBEacon:COMid:SYMBol
[SOURce<HW>]:[ILS]:MBEacon:COMid:TSCHema
[SOURce<HW>]:[ILS]:MBEacon:COMid:[STATe]
```

**class ComidCls**

Comid commands group definition. 11 total commands, 1 Subgroups, 9 group commands

**get_dash**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:DASH
value: float = driver.source.ils.mbeacon.comid.get_dash()
```

Sets the length of a Morse code dash.

**return**
dash: float Range: 0.05 to 1

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:DEPTh
value: float = driver.source.ils.mbeacon.comid.get_depth()
```

Sets the AM modulation depth of the COM/ID signal.

**return**
depth: float Range: 0 to 100

**get_dot**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:DOT
value: float = driver.source.ils.mbeacon.comid.get_dot()
```

Sets the length of a Morse code dot.

**return**
dot: float Range: 0.05 to 1

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:FREQuency
value: float = driver.source.ils.mbeacon.comid.get_frequency()
```

Sets the frequency of the COM/ID signal.

**return**
frequency: float Range: 0.1 to 20E3

**get_letter**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:LETTer
value: float = driver.source.ils.mbeacon.comid.get_letter()
```

Sets the length of a Morse code letter space.

**return**
letter: float Range: 0.05 to 1

**get_period**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:PERiod
value: float = driver.source.ils.mbeacon.comid.get_period()
```

Sets the period of the COM/ID signal.

**return**
period: float Range: 0 to 120

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:[STATe]
value: bool = driver.source.ils.mbeacon.comid.get_state()
```

Enables/disables the COM/ID signal.

**return**
state: 1| ON| 0| OFF

**get_symbol**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:SYMBol
value: float = driver.source.ils.mbeacon.comid.get_symbol()
```

Sets the length of the Morse code symbol space.

**return**
symbol: float Range: 0.05 to 1

**get_tschema**() → AvionicComIdTimeSchem

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:TSCHema
value: enums.AvionicComIdTimeSchem = driver.source.ils.mbeacon.comid.get_
↪tschema()
```

Sets the time schema of the Morse code for the COM/ID signal.

**return**
tschema: STD| USER STD Activates the standard time schema of the Morse code. The set dot length determines the dash length, which is 3 times the dot length. USER Activates the user-defined time schema of the Morse code. Dot and dash length, as well as symbol and letter space can be set separately.

**set_dash**(*dash: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:DASH
driver.source.ils.mbeacon.comid.set_dash(dash = 1.0)
```

Sets the length of a Morse code dash.

**param dash**
float Range: 0.05 to 1

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:DEPTh
driver.source.ils.mbeacon.comid.set_depth(depth = 1.0)
```

Sets the AM modulation depth of the COM/ID signal.

**param depth**
float Range: 0 to 100

**set_dot**(*dot: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:DOT
driver.source.ils.mbeacon.comid.set_dot(dot = 1.0)
```

Sets the length of a Morse code dot.

**param dot**
float Range: 0.05 to 1

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:FREQuency
driver.source.ils.mbeacon.comid.set_frequency(frequency = 1.0)
```

Sets the frequency of the COM/ID signal.

**param frequency**
float Range: 0.1 to 20E3

**set_letter**(*letter: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:LETTer
driver.source.ils.mbeacon.comid.set_letter(letter = 1.0)
```

Sets the length of a Morse code letter space.

**param letter**
float Range: 0.05 to 1

**set_period**(*period: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:PERiod
driver.source.ils.mbeacon.comid.set_period(period = 1.0)
```

Sets the period of the COM/ID signal.

**param period**
float Range: 0 to 120

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:[STATe]
driver.source.ils.mbeacon.comid.set_state(state = False)
```

Enables/disables the COM/ID signal.

**param state**
1| ON| 0| OFF

**set_symbol**(*symbol: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:SYMBol
driver.source.ils.mbeacon.comid.set_symbol(symbol = 1.0)
```

Sets the length of the Morse code symbol space.

**param symbol**
float Range: 0.05 to 1

**set_tschema**(*tschema: AvionicComIdTimeSchem*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:TSCHema
driver.source.ils.mbeacon.comid.set_tschema(tschema = enums.
↪AvionicComIdTimeSchem.STD)
```

Sets the time schema of the Morse code for the COM/ID signal.

**param tschema**
STD| USER STD Activates the standard time schema of the Morse code. The set dot length determines the dash length, which is 3 times the dot length. USER Activates the user-defined time schema of the Morse code. Dot and dash length, as well as symbol and letter space can be set separately.

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.ils.mbeacon.comid.clone()
```

### Subgroups

#### 6.18.11.4.1.1 Code

### SCPI Commands :

```
[SOURce<HW>]:[ILS]:MBEacon:COMid:CODE:STATe
[SOURce<HW>]:[ILS]:MBEacon:COMid:CODE
```

**class CodeCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:CODE:STATe
value: bool = driver.source.ils.mbeacon.comid.code.get_state()
```

No command help available

**return**
state: No help available

**get_value**() → str

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:CODE
value: str = driver.source.ils.mbeacon.comid.code.get_value()
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**return**
code: string

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:CODE:STATe
driver.source.ils.mbeacon.comid.code.set_state(state = False)
```

No command help available

**param state**
No help available

**set_value**(*code: str*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:COMid:CODE
driver.source.ils.mbeacon.comid.code.set_value(code = 'abc')
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**param code**
string

#### 6.18.11.4.2 Frequency

**SCPI Commands :**

```
[SOURce<HW>]:[ILS]:MBEacon:FREQuency:MODE
[SOURce<HW>]:[ILS]:MBEacon:FREQuency
```

**class FrequencyCls**

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

**get_mode**() → AvionicCarrFreqModeMrkBcn

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:FREQuency:MODE
value: enums.AvionicCarrFreqModeMrkBcn = driver.source.ils.mbeacon.frequency.
↪get_mode()
```

Sets the carrier frequency mode of the ILS marker beacon signal.

**return**
mode: USER| PREDefined

**get_value**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:FREQuency
value: float = driver.source.ils.mbeacon.frequency.get_value()
```

Sets the carrier frequency for the ILS marker beacon signal.

**return**
carrier_freq: float Range: 100E3 to 6E9

**set_mode**(*mode: AvionicCarrFreqModeMrkBcn*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:FREQuency:MODE
driver.source.ils.mbeacon.frequency.set_mode(mode = enums.
↪AvionicCarrFreqModeMrkBcn.PREDefined)
```

Sets the carrier frequency mode of the ILS marker beacon signal.

**param mode**
USER| PREDefined

**set_value**(*carrier_freq: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:FREQuency
driver.source.ils.mbeacon.frequency.set_value(carrier_freq = 1.0)
```

Sets the carrier frequency for the ILS marker beacon signal.

**param carrier_freq**
float Range: 100E3 to 6E9

#### 6.18.11.4.3 Marker

**SCPI Commands :**

```
[SOURce<HW>]:[ILS]:MBEacon:MARKer:FREQuency
[SOURce<HW>]:[ILS]:MBEacon:[MARKer]:DEPTh
[SOURce<HW>]:[ILS]:MBEacon:[MARKer]:PULSed
```

**class MarkerCls**

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

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:[MARKer]:DEPTh
value: float = driver.source.ils.mbeacon.marker.get_depth()
```

Sets the modulation depth of the marker signal for the ILS marker beacon signal.

**return**
depth: float Range: 0 to 100

**get_frequency**() → int

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:MARKer:FREQuency
value: int = driver.source.ils.mbeacon.marker.get_frequency()
```

Sets the modulation frequency of the marker signal for the ILS marker beacon modulation signal.

**return**
frequency: 400| 1300| 3000 Unit: Hz

**get_pulsed**() → bool

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:[MARKer]:PULSed
value: bool = driver.source.ils.mbeacon.marker.get_pulsed()
```

Activates the modulation of a pulsed marker signal (morse coding) .

**return**
pulsed: 1| ON| 0| OFF

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:[MARKer]:DEPTh
driver.source.ils.mbeacon.marker.set_depth(depth = 1.0)
```

Sets the modulation depth of the marker signal for the ILS marker beacon signal.

**param depth**
float Range: 0 to 100

**set_frequency**(*frequency: int*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:MARKer:FREQuency
driver.source.ils.mbeacon.marker.set_frequency(frequency = 1)
```

Sets the modulation frequency of the marker signal for the ILS marker beacon modulation signal.

**param frequency**
400| 1300| 3000 Unit: Hz

**set_pulsed**(*pulsed: bool*) → None

```
# SCPI: [SOURce<HW>]:[ILS]:MBEacon:[MARKer]:PULSed
driver.source.ils.mbeacon.marker.set_pulsed(pulsed = False)
```

Activates the modulation of a pulsed marker signal (morse coding) .

**param pulsed**
1| ON| 0| OFF

#### 6.18.11.5 Setting

**SCPI Commands :**

```
[SOURce<HW>]:ILS:SETTing:CATalog
[SOURce<HW>]:ILS:SETTing:DELete
[SOURce<HW>]:ILS:SETTing:LOAD
[SOURce<HW>]:ILS:SETTing:STORe
```

**class SettingCls**

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

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:ILS:SETTing:DELete
driver.source.ils.setting.delete(filename = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path; file extension can be omitted

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:ILS:SETTing:CATalog
value: List[str] = driver.source.ils.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**return**
avionic_ils_cat_names: filename1,filename2,... Returns a string of filenames separated by commas.

**load**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:ILS:SETTing:LOAD
driver.source.ils.setting.load(filename = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path; file extension can be omitted

**set_store**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:ILS:SETTing:STORe
driver.source.ils.setting.set_store(filename = 'abc')
```

Saves the current settings into the selected file; the file extension (*.adf/*.ils/*.vor) is assigned automatically. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path

## 6.18.12 InputPy

**class InputPyCls**

InputPy commands group definition. 3 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.18.12.1 Modext

**class ModextCls**

Modext 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.source.inputPy.modext.clone()
```

### Subgroups

#### 6.18.12.1.1 Coupling<InputIx>

### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.inputPy.modext.coupling.repcap_inputIx_get()
driver.source.inputPy.modext.coupling.repcap_inputIx_set(repcap.InputIx.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:INPut:MODext:COUPling<CH>
```

**class CouplingCls**

Coupling commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: InputIx, default value after init: InputIx.Nr1

**get**(*inputIx=InputIx.Default*) → AcDc

```
# SCPI: [SOURce<HW>]:INPut:MODext:COUPling<CH>
value: enums.AcDc = driver.source.inputPy.modext.coupling.get(inputIx = repcap.
↪InputIx.Default)
```

Selects the coupling mode for an externally applied modulation signal.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Coupling')

**return**
coupling: AC| DC AC Passes the AC signal component of the modulation signal. DC Passes the modulation signal with both components, AC and DC. For active external exponential AM, automatically sets [:SOURcehw]:INPut:MODext:COUPlingchDC.

**set**(*coupling: AcDc*, *inputIx=InputIx.Default*) → None

```
# SCPI: [SOURce<HW>]:INPut:MODext:COUPling<CH>
driver.source.inputPy.modext.coupling.set(coupling = enums.AcDc.AC, inputIx =␣
↪repcap.InputIx.Default)
```

Selects the coupling mode for an externally applied modulation signal.

**param coupling**
AC| DC AC Passes the AC signal component of the modulation signal. DC Passes the modulation signal with both components, AC and DC. For active external exponential AM, automatically sets [:SOURcehw]:INPut:MODext:COUPlingchDC.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Coupling')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.inputPy.modext.coupling.clone()
```

#### 6.18.12.1.2 Impedance<InputIx>

#### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.inputPy.modext.impedance.repcap_inputIx_get()
driver.source.inputPy.modext.impedance.repcap_inputIx_set(repcap.InputIx.Nr1)
```

#### SCPI Command :

```
[SOURce<HW>]:INPut:MODext:IMPedance<CH>
```

**class ImpedanceCls**

Impedance commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: InputIx, default value after init: InputIx.Nr1

**get**(*inputIx=InputIx.Default*) → Imp

```
# SCPI: [SOURce<HW>]:INPut:MODext:IMPedance<CH>
value: enums.Imp = driver.source.inputPy.modext.impedance.get(inputIx = repcap.
↪InputIx.Default)
```

Sets the impedance for the externally supplied modulation signal.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Impedance')

**return**
impedance: G50| G600| HIGH G50 = 50 Ohm to ground G600 = 600 Ohm to ground HIGH = 100 kOhm to ground

**set**(*impedance: Imp*, *inputIx=InputIx.Default*) → None

```
# SCPI: [SOURce<HW>]:INPut:MODext:IMPedance<CH>
driver.source.inputPy.modext.impedance.set(impedance = enums.Imp.G50, inputIx =␣
↪repcap.InputIx.Default)
```

Sets the impedance for the externally supplied modulation signal.

**param impedance**
G50| G600| HIGH G50 = 50 Ohm to ground G600 = 600 Ohm to ground HIGH = 100 kOhm to ground

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Impedance')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.inputPy.modext.impedance.clone()
```

### 6.18.12.2 Trigger

#### SCPI Command :

```
[SOURce]:INPut:TRIGger:SLOPe
```

**class TriggerCls**

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

**get_slope**() → SlopeType

```
# SCPI: [SOURce]:INPut:TRIGger:SLOPe
value: enums.SlopeType = driver.source.inputPy.trigger.get_slope()
```

Sets the polarity of the active slope of an applied instrument trigger.

**return**
slope: NEGative| POSitive

**set_slope**(*slope: SlopeType*) → None

```
# SCPI: [SOURce]:INPut:TRIGger:SLOPe
driver.source.inputPy.trigger.set_slope(slope = enums.SlopeType.NEGative)
```

Sets the polarity of the active slope of an applied instrument trigger.

**param slope**
NEGative| POSitive

## 6.18.13 LffSweep

**class LffSweepCls**

LffSweep 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.source.lffSweep.clone()
```

### Subgroups

#### 6.18.13.1 Trigger

**class TriggerCls**

Trigger 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.source.lffSweep.trigger.clone()
```

### Subgroups

#### 6.18.13.1.1 Source

### SCPI Command :

```
[SOURce<HW>]:LFFSweep:TRIGger:SOURce:ADVanced
```

**class SourceCls**

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

**get_advanced**() → TrigSweepImmBusExt

```
# SCPI: [SOURce<HW>]:LFFSweep:TRIGger:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.source.lffSweep.trigger.source.get_
↪advanced()
```

No command help available

**return**
lf_trig_source_adv: No help available

**set_advanced**(*lf_trig_source_adv: TrigSweepImmBusExt*) → None

```
# SCPI: [SOURce<HW>]:LFFSweep:TRIGger:SOURce:ADVanced
driver.source.lffSweep.trigger.source.set_advanced(lf_trig_source_adv = enums.
↪TrigSweepImmBusExt.BUS)
```

No command help available

**param lf_trig_source_adv**
No help available

### 6.18.14 LfOutput<LfOutput>

#### RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.source.lfOutput.repcap_lfOutput_get()
driver.source.lfOutput.repcap_lfOutput_set(repcap.LfOutput.Nr1)
```

#### SCPI Commands :

```
[SOURce]:LFOutput:OFFSet
[SOURce]:LFOutput:VOLTage
```

**class LfOutputCls**

LfOutput commands group definition. 34 total commands, 8 Subgroups, 2 group commands Repeated Capability: LfOutput, default value after init: LfOutput.Nr1

**get_offset**() → float

```
# SCPI: [SOURce]:LFOutput:OFFSet
value: float = driver.source.lfOutput.get_offset()
```

Sets a DC offset at the LF Output.

**return**
offset: float Range: depends on lfo voltage

**get_voltage**() → float

```
# SCPI: [SOURce]:LFOutput:VOLTage
value: float = driver.source.lfOutput.get_voltage()
```

Sets the voltage of the LF output.

**return**
voltage: float Range: dynamic (see data sheet)

**set_offset**(*offset: float*) → None

```
# SCPI: [SOURce]:LFOutput:OFFSet
driver.source.lfOutput.set_offset(offset = 1.0)
```

Sets a DC offset at the LF Output.

**param offset**
float Range: depends on lfo voltage

**set_voltage**(*voltage: float*) → None

```
# SCPI: [SOURce]:LFOutput:VOLTage
driver.source.lfOutput.set_voltage(voltage = 1.0)
```

Sets the voltage of the LF output.

**param voltage**
float Range: dynamic (see data sheet)

#### Cloning the Group

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

#### Subgroups

##### 6.18.14.1 Bandwidth

#### SCPI Command :

```
[SOURce]:LFOutput<CH>:BANDwidth
```

**class BandwidthCls**

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

**get**(*lfOutput=LfOutput.Default*) → LfBwidth

```
# SCPI: [SOURce]:LFOutput<CH>:BANDwidth
value: enums.LfBwidth = driver.source.lfOutput.bandwidth.get(lfOutput = repcap.
↪LfOutput.Default)
```

Queries the bandwidth of the external LF signal.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
bandwidth: BW0M2| BW10m

##### 6.18.14.2 Frequency

#### SCPI Commands :

```
[SOURce]:LFOutput<CH>:FREQuency
[SOURce<HW>]:LFOutput:FREQuency:MANual
[SOURce<HW>]:LFOutput:FREQuency:MODE
[SOURce<HW>]:LFOutput:FREQuency:STARt
[SOURce<HW>]:LFOutput:FREQuency:STOP
```

**class FrequencyCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce]:LFOutput<CH>:FREQuency
value: float = driver.source.lfOutput.frequency.get(lfOutput = repcap.LfOutput.
↪Default)
```

**Sets the frequency of the LF signal in [:SOURce<hw>]:LFOutput:FREQuency:MODE CW|FIXed mode.**
INTRO_CMD_HELP: Note

- If signal source 'Internal' is set, the instrument performs the analog modulations (AM/FM/PhiM/PM) with this frequency.
- In sweep mode ([:SOURce<hw>]:LFOutput:FREQuency:MODE SWE) , the frequency is coupled with the sweep frequency.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
frequency: float Range: depends on the installed options , Unit: Hz

`get_manual()` → float

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:MANual
value: float = driver.source.lfOutput.frequency.get_manual()
```

Sets the frequency of the subsequent sweep step if LFO:SWE:MODE MAN. Use a separate command for each sweep step.

**return**
manual: float You can select any value within the setting range, where: STARt is set with [:SOURcehw]:LFOutput:FREQuency:STARt STOP is set with [:SOURcehw]:LFOutput:FREQuency:STOP Range: STARt to STOP

`get_mode()` → LfFreqMode

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:MODE
value: enums.LfFreqMode = driver.source.lfOutput.frequency.get_mode()
```

Sets the mode for the output of the LF generator frequency, and determines the commands to be used for frequency settings.

**return**
mode: CW| FIXed| SWEep CW|FIXed Sets the fixed-frequency mode. CW and FIXed are synonyms. To set the output frequency, use command [:SOURce]:LFOutputch:FREQuency SWEep Sets sweep mode. To set the frequency, use the commands: [:SOURcehw]:LFOutput:FREQuency:STARt and [:SOURcehw]:LFOutput:FREQuency:STOP Or [:SOURcehw]:LFOutput:FREQuency:MANual

`get_start()` → float

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:STARt
value: float = driver.source.lfOutput.frequency.get_start()
```

Sets the start/stop frequency for [:SOURce<hw>]:LFOutput:FREQuency:MODE SWEep.

**return**
start: float Range: 0.1 Hz to 1 MHz

`get_stop()` → float

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:STOP
value: float = driver.source.lfOutput.frequency.get_stop()
```

Sets the start/stop frequency for [:SOURce<hw>]:LFOutput:FREQuency:MODE SWEep.

**return**
stop: No help available

**set**(*frequency: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce]:LFOutput<CH>:FREQuency
driver.source.lfOutput.frequency.set(frequency = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

**Sets the frequency of the LF signal in [:SOURce<hw>]:LFOutput:FREQuency:MODE CW|FIXed mode.**
INTRO_CMD_HELP: Note

- If signal source 'Internal' is set, the instrument performs the analog modulations (AM/FM/PhiM/PM) with this frequency.
- In sweep mode ([:SOURce<hw>]:LFOutput:FREQuency:MODE SWE) , the frequency is coupled with the sweep frequency.

**param frequency**
float Range: depends on the installed options , Unit: Hz

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**set_manual**(*manual: float*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:MANual
driver.source.lfOutput.frequency.set_manual(manual = 1.0)
```

Sets the frequency of the subsequent sweep step if LFO:SWE:MODE MAN. Use a separate command for each sweep step.

**param manual**
float You can select any value within the setting range, where: STARt is set with [:SOURcehw]:LFOutput:FREQuency:STARt STOP is set with [:SOURcehw]:LFOutput:FREQuency:STOP Range: STARt to STOP

**set_mode**(*mode: LfFreqMode*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:MODE
driver.source.lfOutput.frequency.set_mode(mode = enums.LfFreqMode.CW)
```

Sets the mode for the output of the LF generator frequency, and determines the commands to be used for frequency settings.

**param mode**
CW| FIXed| SWEep CW|FIXed Sets the fixed-frequency mode. CW and FIXed are synonyms. To set the output frequency, use command [:SOURce]:LFOutputch:FREQuency SWEep Sets sweep mode. To set the frequency, use the commands: [:SOURcehw]:LFOutput:FREQuency:STARt and [:SOURcehw]:LFOutput:FREQuency:STOP Or [:SOURcehw]:LFOutput:FREQuency:MANual

**set_start**(*start: float*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:STARt
driver.source.lfOutput.frequency.set_start(start = 1.0)
```

Sets the start/stop frequency for [:SOURce<hw>]:LFOutput:FREQuency:MODE SWEep.

**param start**
float Range: 0.1 Hz to 1 MHz

`set_stop`(*stop: float*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:FREQuency:STOP
driver.source.lfOutput.frequency.set_stop(stop = 1.0)
```

Sets the start/stop frequency for [:SOURce<hw>]:LFOutput:FREQuency:MODE SWEep.

**param stop**
float Range: 0.1 Hz to 1 MHz

#### 6.18.14.3 Internal

`class InternalCls`

Internal 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.source.lfOutput.internal.clone()
```

##### Subgroups

###### 6.18.14.3.1 Voltage

##### SCPI Command :

```
[SOURce]:LFOutput<CH>:INTernal:VOLTage
```

`class VoltageCls`

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

`get`(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce]:LFOutput<CH>:INTernal:VOLTage
value: float = driver.source.lfOutput.internal.voltage.get(lfOutput = repcap.
↪LfOutput.Default)
```

Sets the output voltage for the LF generators. The sum of both values must not exceed the overall output voltage, set with command [:SOURce]:LFOutput:VOLTage.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
voltage: float Range: 0 to 4

**set**(*voltage: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce]:LFOutput<CH>:INTernal:VOLTage
driver.source.lfOutput.internal.voltage.set(voltage = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Sets the output voltage for the LF generators. The sum of both values must not exceed the overall output voltage, set with command [:SOURce]:LFOutput:VOLTage.

**param voltage**
float Range: 0 to 4

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.4 Period

**SCPI Command :**

```
[SOURce<HW>]:LFOutput<CH>:PERiod
```

**class PeriodCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:PERiod
value: float = driver.source.lfOutput.period.get(lfOutput = repcap.LfOutput.
↪Default)
```

Queries the repetition frequency of the sine signal.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
lf_sine_period: float Range: 1E-6 to 100, Unit: s

#### 6.18.14.5 Shape

**SCPI Command :**

```
[SOURce<HW>]:LFOutput<CH>:SHAPe
```

**class ShapeCls**

Shape commands group definition. 10 total commands, 3 Subgroups, 1 group commands

**get**(*lfOutput=LfOutput.Default*) → LfShapeBfAmily

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe
value: enums.LfShapeBfAmily = driver.source.lfOutput.shape.get(lfOutput =␣
↪repcap.LfOutput.Default)
```

Selects the waveform shape of the LF signal.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
shape: SINE| SQUare| PULSe| TRIangle| TRAPeze

**set**(*shape: LfShapeBfAmily*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe
driver.source.lfOutput.shape.set(shape = enums.LfShapeBfAmily.PULSe, lfOutput =␣
↪repcap.LfOutput.Default)
```

Selects the waveform shape of the LF signal.

**param shape**
SINE| SQUare| PULSe| TRIangle| TRAPeze

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

### Cloning the Group

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

### Subgroups

#### 6.18.14.5.1 Pulse

**class PulseCls**

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

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.lfOutput.shape.pulse.clone()
```

### Subgroups

#### 6.18.14.5.1.1 Dcycle

### SCPI Command :

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:DCYCle
```

**class DcycleCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:DCYCle
value: float = driver.source.lfOutput.shape.pulse.dcycle.get(lfOutput = repcap.
↪LfOutput.Default)
```

Sets the duty cycle for the shape pulse.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
dcycle: float Range: 1E-6 to 100, Unit: PCT

**set**(*dcycle: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:DCYCle
driver.source.lfOutput.shape.pulse.dcycle.set(dcycle = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Sets the duty cycle for the shape pulse.

**param dcycle**
float Range: 1E-6 to 100, Unit: PCT

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.1.2 Period

### SCPI Command :

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:PERiod
```

**class PeriodCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:PERiod
value: float = driver.source.lfOutput.shape.pulse.period.get(lfOutput = repcap.
↪LfOutput.Default)
```

Sets the period of the generated pulse. The period determines the repetition frequency of the internal signal.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
period: float Range: 1E-6 to 100

**set**(*period: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:PERiod
driver.source.lfOutput.shape.pulse.period.set(period = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Sets the period of the generated pulse. The period determines the repetition frequency of the internal signal.

**param period**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.1.3 Width

#### SCPI Command :

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:WIDTh
```

**class WidthCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:WIDTh
value: float = driver.source.lfOutput.shape.pulse.width.get(lfOutput = repcap.
↪LfOutput.Default)
```

Sets the pulse width of the generated pulse.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
width: float Range: 1E-6 to 100

**set**(*width: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:PULSe:WIDTh
driver.source.lfOutput.shape.pulse.width.set(width = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Sets the pulse width of the generated pulse.

**param width**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.2 Trapeze

**class TrapezeCls**

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

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.lfOutput.shape.trapeze.clone()
```

### Subgroups

#### 6.18.14.5.2.1 Fall

### SCPI Command :

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:FALL
```

**class FallCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:FALL
value: float = driver.source.lfOutput.shape.trapeze.fall.get(lfOutput = repcap.
↪LfOutput.Default)
```

Selects the fall time for the trapezoid shape of the LF generator.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
fall: float Range: 1E-6 to 100

**set**(*fall: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:FALL
driver.source.lfOutput.shape.trapeze.fall.set(fall = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Selects the fall time for the trapezoid shape of the LF generator.

**param fall**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.2.2 High

**SCPI Command :**

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:HIGH
```

**class HighCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:HIGH
value: float = driver.source.lfOutput.shape.trapeze.high.get(lfOutput = repcap.
↪LfOutput.Default)
```

Sets the high time for the trapezoid signal of the LF generator.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
high: float Range: 1E-6 to 100

**set**(*high: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:HIGH
driver.source.lfOutput.shape.trapeze.high.set(high = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Sets the high time for the trapezoid signal of the LF generator.

**param high**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.2.3 Period

**SCPI Command :**

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:PERiod
```

**class PeriodCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:PERiod
value: float = driver.source.lfOutput.shape.trapeze.period.get(lfOutput =␣
↪repcap.LfOutput.Default)
```

Sets the period of the generated trapezoid shape. The period determines the repetition frequency of the internal signal.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
period: float Range: 1E-6 to 100

**set**(*period: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:PERiod
driver.source.lfOutput.shape.trapeze.period.set(period = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Sets the period of the generated trapezoid shape. The period determines the repetition frequency of the internal signal.

**param period**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.2.4 Rise

**SCPI Command :**

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:RISE
```

**class RiseCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:RISE
value: float = driver.source.lfOutput.shape.trapeze.rise.get(lfOutput = repcap.
↪LfOutput.Default)
```

Selects the rise time for the trapezoid shape of the LF generator.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
rise: float Range: 1E-6 to 100

**set**(*rise: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRAPeze:RISE
driver.source.lfOutput.shape.trapeze.rise.set(rise = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Selects the rise time for the trapezoid shape of the LF generator.

**param rise**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.3 Triangle

**class TriangleCls**

Triangle 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.source.lfOutput.shape.triangle.clone()
```

### Subgroups

#### 6.18.14.5.3.1 Period

### SCPI Command :

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:TRIangle:PERiod
```

**class PeriodCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRIangle:PERiod
value: float = driver.source.lfOutput.shape.triangle.period.get(lfOutput =␣
↪repcap.LfOutput.Default)
```

Sets the period of the generated pulse. The period determines the repetition frequency of the internal signal.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
period: float Range: 1E-6 to 100

**set**(*period: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRIangle:PERiod
driver.source.lfOutput.shape.triangle.period.set(period = 1.0, lfOutput =␣
↪repcap.LfOutput.Default)
```

Sets the period of the generated pulse. The period determines the repetition frequency of the internal signal.

**param period**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.5.3.2 Rise

**SCPI Command :**

```
[SOURce<HW>]:LFOutput<CH>:SHAPe:TRIangle:RISE
```

**class RiseCls**

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

**get**(*lfOutput=LfOutput.Default*) → float

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRIangle:RISE
value: float = driver.source.lfOutput.shape.triangle.rise.get(lfOutput = repcap.
↪LfOutput.Default)
```

Selects the rise time for the triangle single of the LF generator.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
rise: float Range: 1E-6 to 100

**set**(*rise: float*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce<HW>]:LFOutput<CH>:SHAPe:TRIangle:RISE
driver.source.lfOutput.shape.triangle.rise.set(rise = 1.0, lfOutput = repcap.
↪LfOutput.Default)
```

Selects the rise time for the triangle single of the LF generator.

**param rise**
float Range: 1E-6 to 100

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.6 Source

**SCPI Command :**

```
[SOURce]:LFOutput<CH>:SOURce
```

**class SourceCls**

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

**get**(*lfOutput=LfOutput.Default*) → LfSource

```
# SCPI: [SOURce]:LFOutput<CH>:SOURce
value: enums.LfSource = driver.source.lfOutput.source.get(lfOutput = repcap.
↪LfOutput.Default)
```

Determines the LF signal to be synchronized, when monitoring is enabled.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
source: LF1| LF2| NOISe| AM| FMPM| EXT1 | | EXT2 LF1|LF2 Selects an internally generated LF signal. NOISe Selects an internally generated noise signal. EXT1|EXT2 Selects an externally supplied LF signal AM Selects the AM signal. FMPM Selects the signal also used by the frequency or phase modulations.

**set**(*source: LfSource*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce]:LFOutput<CH>:SOURce
driver.source.lfOutput.source.set(source = enums.LfSource.AM, lfOutput = repcap.
↪LfOutput.Default)
```

Determines the LF signal to be synchronized, when monitoring is enabled.

**param source**
LF1| LF2| NOISe| AM| FMPM| EXT1 | | EXT2 LF1|LF2 Selects an internally generated LF signal. NOISe Selects an internally generated noise signal. EXT1|EXT2 Selects an externally supplied LF signal AM Selects the AM signal. FMPM Selects the signal also used by the frequency or phase modulations.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.7 State

**SCPI Command :**

```
[SOURce]:LFOutput<CH>:[STATe]
```

**class StateCls**

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

**get**(*lfOutput=LfOutput.Default*) → bool

```
# SCPI: [SOURce]:LFOutput<CH>:[STATe]
value: bool = driver.source.lfOutput.state.get(lfOutput = repcap.LfOutput.
↪Default)
```

Activates LF signal output.

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *lfOutput=LfOutput.Default*) → None

```
# SCPI: [SOURce]:LFOutput<CH>:[STATe]
driver.source.lfOutput.state.set(state = False, lfOutput = repcap.LfOutput.
↪Default)
```

Activates LF signal output.

**param state**
1| ON| 0| OFF

**param lfOutput**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'LfOutput')

#### 6.18.14.8 Sweep

**class SweepCls**

Sweep commands group definition. 12 total commands, 1 Subgroups, 0 group commands

##### Cloning the Group

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

##### Subgroups

###### 6.18.14.8.1 Frequency

##### SCPI Commands :

```
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:DWELl
[SOURce]:LFOutput:SWEep:[FREQuency]:LFSource
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:POINts
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:RETRace
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:RUNNing
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:SHAPe
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:SPACing
```

**class FrequencyCls**

Frequency commands group definition. 12 total commands, 3 Subgroups, 7 group commands

**get_dwell**() → float

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:DWELl
value: float = driver.source.lfOutput.sweep.frequency.get_dwell()
```

Sets the dwell time for each frequency step of the sweep.

**return**
dwell: float Range: 0.001 to 100, Unit: s

**get_lf_source**() → LfSweepSource

```
# SCPI: [SOURce]:LFOutput:SWEep:[FREQuency]:LFSource
value: enums.LfSweepSource = driver.source.lfOutput.sweep.frequency.get_lf_
↪source()
```

No command help available

**return**
lf_source: No help available

**get_points**() → int

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:POINts
value: int = driver.source.lfOutput.sweep.frequency.get_points()
```

Sets the number of steps in an LF sweep. For information on how the value is calculated and the interdependency with other parameters, see 'Correlating parameters in sweep mode'

**return**
points: integer Range: 2 to POINts

**get_retrace**() → bool

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:RETRace
value: bool = driver.source.lfOutput.sweep.frequency.get_retrace()
```

Activates that the signal changes to the start frequency value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**return**
state: 1| ON| 0| OFF

**get_running**() → bool

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:RUNNing
value: bool = driver.source.lfOutput.sweep.frequency.get_running()
```

Queries the current status of the LF frequency sweep mode.

**return**
state: 1| ON| 0| OFF

**get_shape**() → SweCyclMode

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:SHAPe
value: enums.SweCyclMode = driver.source.lfOutput.sweep.frequency.get_shape()
```

Sets the cycle mode for a sweep sequence (shape) .

**return**
shape: SAWTooth| TRIangle

**get_spacing**() → Spacing

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:SPACing
value: enums.Spacing = driver.source.lfOutput.sweep.frequency.get_spacing()
```

Selects linear or logarithmic sweep spacing.

**return**
spacing: LINear| LOGarithmic

**set_dwell**(*dwell: float*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:DWELl
driver.source.lfOutput.sweep.frequency.set_dwell(dwell = 1.0)
```

Sets the dwell time for each frequency step of the sweep.

**param dwell**
float Range: 0.001 to 100, Unit: s

**set_lf_source**(*lf_source: LfSweepSource*) → None

```
# SCPI: [SOURce]:LFOutput:SWEep:[FREQuency]:LFSource
driver.source.lfOutput.sweep.frequency.set_lf_source(lf_source = enums.
↪LfSweepSource.LF1)
```

No command help available

**param lf_source**
No help available

**set_points**(*points: int*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:POINts
driver.source.lfOutput.sweep.frequency.set_points(points = 1)
```

Sets the number of steps in an LF sweep. For information on how the value is calculated and the interdependency with other parameters, see 'Correlating parameters in sweep mode'

**param points**
integer Range: 2 to POINts

**set_retrace**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:RETRace
driver.source.lfOutput.sweep.frequency.set_retrace(state = False)
```

Activates that the signal changes to the start frequency value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**param state**
1| ON| 0| OFF

**set_shape**(*shape: SweCyclMode*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:SHAPe
driver.source.lfOutput.sweep.frequency.set_shape(shape = enums.SweCyclMode.
↪SAWTooth)
```

Sets the cycle mode for a sweep sequence (shape) .

**param shape**
SAWTooth| TRIangle

**set_spacing**(*spacing: Spacing*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:SPACing
driver.source.lfOutput.sweep.frequency.set_spacing(spacing = enums.Spacing.
↪LINear)
```

Selects linear or logarithmic sweep spacing.

**param spacing**
LINear| LOGarithmic

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.lfOutput.sweep.frequency.clone()
```

### Subgroups

#### 6.18.14.8.1.1 Execute

### SCPI Command :

```
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:EXECute
driver.source.lfOutput.sweep.frequency.execute.set()
```

Immediately starts an LF sweep. [:SOURce<hw>]:LFOutput:SWEep[:FREQuency]:MODE determines which sweep is executed, e.g. SOURce:LFOutput:SWEep:FREQuency:MODE STEP.

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

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:EXECute
driver.source.lfOutput.sweep.frequency.execute.set_with_opc()
```

Immediately starts an LF sweep. [:SOURce<hw>]:LFOutput:SWEep[:FREQuency]:MODE determines which sweep is executed, e.g. SOURce:LFOutput:SWEep:FREQuency:MODE STEP.

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

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

#### 6.18.14.8.1.2 Mode

##### SCPI Commands :

```
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:MODE:ADVanced
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:MODE
```

**class ModeCls**

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

**get_advanced**() → AutoManualMode

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:MODE:ADVanced
value: enums.AutoManualMode = driver.source.lfOutput.sweep.frequency.mode.get_
↪advanced()
```

No command help available

**return**
lf_sweep_mode_adv: No help available

**get_value**() → AutoManStep

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:MODE
value: enums.AutoManStep = driver.source.lfOutput.sweep.frequency.mode.get_
↪value()
```

Sets the cycle mode of the LF sweep.

**return**
mode: AUTO| MANual| STEP AUTO Performs a complete sweep cycle from the start to the end value when a trigger event occurs. The dwell time determines the time period until the signal switches to the next step. MANual Performs a single sweep step when a manual trigger event occurs. The trigger system is not active. To trigger each frequency step of the sweep individually, use the command [:SOURcehw]:LFOutput:FREQuency:MANual. STEP Each trigger command triggers one sweep step only. The frequency increases by the value set with the coammnds: [:SOURcehw]:LFOutput:SWEep[:FREQuency]:STEP[:LINear] (linear spacing) [:SOURcehw]:LFOutput:SWEep[:FREQuency]:STEP:LOGarithmic(logarithmic spacing)

**set_advanced**(*lf_sweep_mode_adv: AutoManualMode*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:MODE:ADVanced
driver.source.lfOutput.sweep.frequency.mode.set_advanced(lf_sweep_mode_adv =␣
↪enums.AutoManualMode.AUTO)
```

No command help available

**param lf_sweep_mode_adv**
No help available

**set_value**(*mode: AutoManStep*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:MODE
driver.source.lfOutput.sweep.frequency.mode.set_value(mode = enums.AutoManStep.
↪AUTO)
```

Sets the cycle mode of the LF sweep.

**param mode**
AUTO| MANual| STEP AUTO Performs a complete sweep cycle from the start to the end value when a trigger event occurs. The dwell time determines the time period until the signal switches to the next step. MANual Performs a single sweep step when a manual trigger event occurs. The trigger system is not active. To trigger each frequency step of the sweep individually, use the command [:SOURcehw]:LFOutput:FREQuency:MANual. STEP Each trigger command triggers one sweep step only. The frequency increases by the value set with the coammnds: [:SOURcehw]:LFOutput:SWEep[:FREQuency]:STEP[:LINear] (linear spacing) [:SOURcehw]:LFOutput:SWEep[:FREQuency]:STEP:LOGarithmic(logarithmic spacing)

#### 6.18.14.8.1.3 Step

**SCPI Commands :**

```
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:STEP:LOGarithmic
[SOURce<HW>]:LFOutput:SWEep:[FREQuency]:STEP:[LINear]
```

**class StepCls**

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

**get_linear**() → float

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:STEP:[LINear]
value: float = driver.source.lfOutput.sweep.frequency.step.get_linear()
```

Sets the step width for the linear sweep. For information on how the value is calculated and the interdependency with other parameters, see 'Correlating parameters in sweep mode'

**return**
linear: float Range: 0.1 to STOP-STARt

**get_logarithmic**() → float

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:STEP:LOGarithmic
value: float = driver.source.lfOutput.sweep.frequency.step.get_logarithmic()
```

Sets the step width factor for logarithmic sweeps to calculate the frequencies of the steps. For information on how the value is calculated and the interdependency with other parameters, see 'Correlating parameters in sweep mode'

**return**
logarithmic: float The unit is mandatory Range: 0.01 to 100, Unit: PCT

**set_linear**(*linear: float*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:STEP:[LINear]
driver.source.lfOutput.sweep.frequency.step.set_linear(linear = 1.0)
```

Sets the step width for the linear sweep. For information on how the value is calculated and the interdependency with other parameters, see 'Correlating parameters in sweep mode'

**param linear**
float Range: 0.1 to STOP-STARt

**set_logarithmic**(*logarithmic: float*) → None

```
# SCPI: [SOURce<HW>]:LFOutput:SWEep:[FREQuency]:STEP:LOGarithmic
driver.source.lfOutput.sweep.frequency.step.set_logarithmic(logarithmic = 1.0)
```

Sets the step width factor for logarithmic sweeps to calculate the frequencies of the steps. For information on how the value is calculated and the interdependency with other parameters, see 'Correlating parameters in sweep mode'

**param logarithmic**
float The unit is mandatory Range: 0.01 to 100, Unit: PCT

### 6.18.15 ListPy

#### SCPI Commands :

```
[SOURce<HW>]:LIST:CATalog
[SOURce<HW>]:LIST:DELete
[SOURce<HW>]:LIST:DELete:ALL
[SOURce<HW>]:LIST:FREE
[SOURce<HW>]:LIST:RESet
[SOURce<HW>]:LIST:RMODe
[SOURce<HW>]:LIST:RUNNing
[SOURce<HW>]:LIST:SELect
```

**class ListPyCls**

ListPy commands group definition. 34 total commands, 8 Subgroups, 8 group commands

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:LIST:DELete
driver.source.listPy.delete(filename = 'abc')
```

Deletes the specified list. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
string Filename or complete file path; file extension is optional.

**delete_all**() → None

```
# SCPI: [SOURce<HW>]:LIST:DELete:ALL
driver.source.listPy.delete_all()
```

**Deletes all lists in the set directory.**
INTRO_CMD_HELP: This command can only be executed, if:

- No list file is selected.
- List mode is disabled.

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

```
# SCPI: [SOURce<HW>]:LIST:DELete:ALL
driver.source.listPy.delete_all_with_opc()
```

**Deletes all lists in the set directory.**
INTRO_CMD_HELP: This command can only be executed, if:

- No list file is selected.
- List mode is disabled.

Same as delete_all, but waits for the operation to complete before continuing further. Use the RsSmab.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**() → List[str]

```
# SCPI: [SOURce<HW>]:LIST:CATalog
value: List[str] = driver.source.listPy.get_catalog()
```

Queries the available list files in the specified directory.

**return**
catalog: string List of list filenames, separated by commas

**get_free**() → int

```
# SCPI: [SOURce<HW>]:LIST:FREE
value: int = driver.source.listPy.get_free()
```

Queries the amount of free memory (in bytes) for list mode lists.

**return**
free: integer Range: 0 to INT_MAX

**get_rmode**() → LmodRunMode

```
# SCPI: [SOURce<HW>]:LIST:RMODe
value: enums.LmodRunMode = driver.source.listPy.get_rmode()
```

No command help available

**return**
rmode: No help available

**get_running**() → bool

```
# SCPI: [SOURce<HW>]:LIST:RUNNing
value: bool = driver.source.listPy.get_running()
```

Queries the current state of the list mode.

**return**
state: 1| ON| 0| OFF 1 Signal generation based on the list mode is active.

**get_select**() → str

```
# SCPI: [SOURce<HW>]:LIST:SELect
value: str = driver.source.listPy.get_select()
```

Selects or creates a data list in list mode. If the list with the selected name does not exist, a new list is created.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**reset**() → None

```
# SCPI: [SOURce<HW>]:LIST:RESet
driver.source.listPy.reset()
```

Jumps to the beginning of the list.

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

```
# SCPI: [SOURce<HW>]:LIST:RESet
driver.source.listPy.reset_with_opc()
```

Jumps to the beginning of the list.

Same as reset, but waits for the operation to complete before continuing further. Use the RsSmab.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_rmode**(*rmode: LmodRunMode*) → None

```
# SCPI: [SOURce<HW>]:LIST:RMODe
driver.source.listPy.set_rmode(rmode = enums.LmodRunMode.LEARned)
```

No command help available

**param rmode**
No help available

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:LIST:SELect
driver.source.listPy.set_select(filename = 'abc')
```

Selects or creates a data list in list mode. If the list with the selected name does not exist, a new list is created.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

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

### Subgroups

#### 6.18.15.1 Dexchange

### SCPI Commands :

```
[SOURce<HW>]:LIST:DEXChange:MODE
[SOURce<HW>]:LIST:DEXChange:SELect
```

**class DexchangeCls**

Dexchange commands group definition. 8 total commands, 2 Subgroups, 2 group commands

**get_mode**() → DexchMode

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:MODE
value: enums.DexchMode = driver.source.listPy.dexchange.get_mode()
```

Determines the import or export of a list. Specify the source or destination file with the command [:SOURce<hw>]:LIST:DEXChange:SELect.

**return**
mode: IMPort| EXPort

**get_select**() → str

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:SELect
value: str = driver.source.listPy.dexchange.get_select()
```

Selects the ASCII file for import or export, containing a list.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_mode**(*mode: DexchMode*) → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:MODE
driver.source.listPy.dexchange.set_mode(mode = enums.DexchMode.EXPort)
```

Determines the import or export of a list. Specify the source or destination file with the command [:SOURce<hw>]:LIST:DEXChange:SELect.

**param mode**
IMPort| EXPort

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:SELect
driver.source.listPy.dexchange.set_select(filename = 'abc')
```

Selects the ASCII file for import or export, containing a list.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

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

### Subgroups

#### 6.18.15.1.1 Afile

### SCPI Commands :

```
[SOURce<HW>]:LIST:DEXChange:AFILe:CATalog
[SOURce<HW>]:LIST:DEXChange:AFILe:EXTension
[SOURce<HW>]:LIST:DEXChange:AFILe:SELect
```

**class AfileCls**

Afile commands group definition. 5 total commands, 1 Subgroups, 3 group commands

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:CATalog
value: List[str] = driver.source.listPy.dexchange.afile.get_catalog()
```

Queries the available ASCII files for export or import of list mode data in the current or specified directory.

**return**
catalog: string List of ASCII files *.txt or *.csv, separated by commas.

**get_extension**() → DexchExtension

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:EXTension
value: enums.DexchExtension = driver.source.listPy.dexchange.afile.get_
↪extension()
```

Determines the extension of the ASCII file for import or export, or to query existing files.

**return**
extension: TXT| CSV

**get_select**() → str

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SELect
value: str = driver.source.listPy.dexchange.afile.get_select()
```

Selects the ASCII file to be imported or exported.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_extension**(*extension: DexchExtension*) → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:EXTension
driver.source.listPy.dexchange.afile.set_extension(extension = enums.
↪DexchExtension.CSV)
```

Determines the extension of the ASCII file for import or export, or to query existing files.

**param extension**
TXT| CSV

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SELect
driver.source.listPy.dexchange.afile.set_select(filename = 'abc')
```

Selects the ASCII file to be imported or exported.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.listPy.dexchange.afile.clone()
```

### Subgroups

#### 6.18.15.1.1.1 Separator

### SCPI Commands :

```
[SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:COLumn
[SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:DECimal
```

**class SeparatorCls**

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

**get_column**() → DexchSepCol

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:COLumn
value: enums.DexchSepCol = driver.source.listPy.dexchange.afile.separator.get_
↪column()
```

Selects the separator between the frequency and level column of the ASCII table.

**return**
column: TABulator| SEMicolon| COMMa| SPACe

**get_decimal**() → DecimalSeparator

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:DECimal
value: enums.DecimalSeparator = driver.source.listPy.dexchange.afile.separator.
↪get_decimal()
```

Sets '.' (decimal point) or ',' (comma) as the decimal separator used in the ASCII data with floating-point numerals.

**return**
decimal: DOT| COMMa

**set_column**(*column: DexchSepCol*) → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:COLumn
driver.source.listPy.dexchange.afile.separator.set_column(column = enums.
↪DexchSepCol.COMMa)
```

Selects the separator between the frequency and level column of the ASCII table.

**param column**
TABulator| SEMicolon| COMMa| SPACe

**set_decimal**(*decimal: DecimalSeparator*) → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:DECimal
driver.source.listPy.dexchange.afile.separator.set_decimal(decimal = enums.
↪DecimalSeparator.COMMa)
```

Sets '.' (decimal point) or ',' (comma) as the decimal separator used in the ASCII data with floating-point numerals.

**param decimal**
DOT| COMMa

#### 6.18.15.1.2 Execute

##### SCPI Command :

```
[SOURce<HW>]:LIST:DEXChange:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:EXECute
driver.source.listPy.dexchange.execute.set()
```

Executes the import or export of the selected list file, according to the previously set transfer direction with command [:SOURce<hw>]:LIST:DEXChange:MODE

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

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:EXECute
driver.source.listPy.dexchange.execute.set_with_opc()
```

Executes the import or export of the selected list file, according to the previously set transfer direction with command [:SOURce<hw>]:LIST:DEXChange:MODE

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

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

#### 6.18.15.2 Dwell

##### SCPI Commands :

```
[SOURce<HW>]:LIST:DWELl:MODE
[SOURce<HW>]:LIST:DWELl
```

**class DwellCls**

Dwell commands group definition. 4 total commands, 1 Subgroups, 2 group commands

**get_mode**() → ParameterSetMode

```
# SCPI: [SOURce<HW>]:LIST:DWELl:MODE
value: enums.ParameterSetMode = driver.source.listPy.dwell.get_mode()
```

Selects the dwell time mode.

**return**
dwell_mode: LIST| GLOBal LIST Uses the dwell time, specified in the data table for each value pair individually. GLOBal Uses a constant dwell time, set with command [:SOURcehw]:LIST:DWELl.

**get_value**() → float

```
# SCPI: [SOURce<HW>]:LIST:DWELl
value: float = driver.source.listPy.dwell.get_value()
```

Sets the global dwell time. The instrument generates the signal with the frequency / power value pairs of each list entry for that particular period. See also 'Significant parameters and functions'.

**return**
dwell: float Range: 1E-3 to 100

**set_mode**(*dwell_mode: ParameterSetMode*) → None

```
# SCPI: [SOURce<HW>]:LIST:DWELl:MODE
driver.source.listPy.dwell.set_mode(dwell_mode = enums.ParameterSetMode.GLOBal)
```

Selects the dwell time mode.

**param dwell_mode**
LIST| GLOBal LIST Uses the dwell time, specified in the data table for each

value pair individually. GLOBal Uses a constant dwell time, set with command [:SOURcehw]:LIST:DWELl.

**set_value**(*dwell: float*) → None

```
# SCPI: [SOURce<HW>]:LIST:DWELl
driver.source.listPy.dwell.set_value(dwell = 1.0)
```

Sets the global dwell time. The instrument generates the signal with the frequency / power value pairs of each list entry for that particular period. See also 'Significant parameters and functions'.

**param dwell**
float Range: 1E-3 to 100

### Cloning the Group

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

### Subgroups

#### 6.18.15.2.1 ListPy

### SCPI Commands :

```
[SOURce<HW>]:LIST:DWELl:LIST:POINts
[SOURce<HW>]:LIST:DWELl:LIST
```

**class ListPyCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:LIST:DWELl:LIST:POINts
value: int = driver.source.listPy.dwell.listPy.get_points()
```

Queries the number (points) of dwell time entries in the selected list.

**return**
points: integer Range: 0 to INT_MAX

**get_value**() → List[int]

```
# SCPI: [SOURce<HW>]:LIST:DWELl:LIST
value: List[int] = driver.source.listPy.dwell.listPy.get_value()
```

Enters the dwell time values in the selected list in us.

**return**
dwell: Dwell#1{, Dwell#2, ...} | block data You can either enter the data as a list of numbers, or as binary block data. The list of numbers can be of any length, with the list entries separated by commas. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See also method RsSmab.FormatPy.data for more details.

**set_value**(*dwell: List[int]*) → None

```
# SCPI: [SOURce<HW>]:LIST:DWELl:LIST
driver.source.listPy.dwell.listPy.set_value(dwell = [1, 2, 3])
```

Enters the dwell time values in the selected list in us.

**param dwell**
Dwell#1{, Dwell#2, ...} | block data You can either enter the data as a list of numbers, or as binary block data. The list of numbers can be of any length, with the list entries separated by commas. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See also method RsSmab.FormatPy.data for more details.

#### 6.18.15.3 Frequency

**SCPI Commands :**

```
[SOURce<HW>]:LIST:FREQuency:POINts
[SOURce<HW>]:LIST:FREQuency
```

**class FrequencyCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:LIST:FREQuency:POINts
value: int = driver.source.listPy.frequency.get_points()
```

Queries the number (points) of frequency entries in the seleced list.

**return**
points: integer Range: 0 to INT_MAX

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:LIST:FREQuency
value: List[float] = driver.source.listPy.frequency.get_value()
```

Enters the frequency values in the selected list.

**return**
frequency: Frequency#1{, Frequency#2, ...} | block data You can either enter the data as a list of numbers, or as binary block data. The list of numbers can be of any length, with the list entries separated by commas. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See also method RsSmab.FormatPy.data. Range: 300 kHz to RFmax (depends on the installed options)

**set_value**(*frequency: List[float]*) → None

```
# SCPI: [SOURce<HW>]:LIST:FREQuency
driver.source.listPy.frequency.set_value(frequency = [1.1, 2.2, 3.3])
```

Enters the frequency values in the selected list.

**param frequency**
Frequency#1{, Frequency#2, ...} | block data You can either enter the data as a list of numbers, or as binary block data. The list of numbers can be of any length, with the list entries separated by commas. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See also method RsSmab.FormatPy.data. Range: 300 kHz to RFmax (depends on the installed options)

#### 6.18.15.4 Index

**SCPI Commands :**

```
[SOURce<HW>]:LIST:INDex:STARt
[SOURce<HW>]:LIST:INDex:STOP
[SOURce<HW>]:LIST:INDex
```

**class IndexCls**

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

**get_start**() → int

```
# SCPI: [SOURce<HW>]:LIST:INDex:STARt
value: int = driver.source.listPy.index.get_start()
```

Sets the start and stop index of the index range which defines a subgroup of frequency/level value pairs in the current list.

**return**
start: No help available

**get_stop**() → int

```
# SCPI: [SOURce<HW>]:LIST:INDex:STOP
value: int = driver.source.listPy.index.get_stop()
```

Sets the start and stop index of the index range which defines a subgroup of frequency/level value pairs in the current list.

**return**
stop: integer Index range Only values inside this range are processed in list mode Range: 0 to list length

**get_value**() → int

```
# SCPI: [SOURce<HW>]:LIST:INDex
value: int = driver.source.listPy.index.get_value()
```

Sets the list index in LIST:MODE STEP. After the trigger signal, the instrument processes the frequency and level settings of the selected index.

**return**
index: integer

**set_start**(*start: int*) → None

```
# SCPI: [SOURce<HW>]:LIST:INDex:STARt
driver.source.listPy.index.set_start(start = 1)
```

Sets the start and stop index of the index range which defines a subgroup of frequency/level value pairs in the current list.

**param start**
integer Index range Only values inside this range are processed in list mode Range: 0 to list length

**set_stop**(*stop: int*) → None

```
# SCPI: [SOURce<HW>]:LIST:INDex:STOP
driver.source.listPy.index.set_stop(stop = 1)
```

Sets the start and stop index of the index range which defines a subgroup of frequency/level value pairs in the current list.

**param stop**
integer Index range Only values inside this range are processed in list mode Range: 0 to list length

**set_value**(*index: int*) → None

```
# SCPI: [SOURce<HW>]:LIST:INDex
driver.source.listPy.index.set_value(index = 1)
```

Sets the list index in LIST:MODE STEP. After the trigger signal, the instrument processes the frequency and level settings of the selected index.

**param index**
integer

#### 6.18.15.5 Learn

**SCPI Command :**

```
[SOURce<HW>]:LIST:LEARn
```

**class LearnCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:LIST:LEARn
driver.source.listPy.learn.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:LIST:LEARn
driver.source.listPy.learn.set_with_opc()
```

No command help available

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

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

#### 6.18.15.6 Mode

**SCPI Commands :**

```
[SOURce<HW>]:LIST:MODE:ADVanced
[SOURce<HW>]:LIST:MODE
```

**class ModeCls**

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

**get_advanced**() → AutoManualMode

```
# SCPI: [SOURce<HW>]:LIST:MODE:ADVanced
value: enums.AutoManualMode = driver.source.listPy.mode.get_advanced()
```

No command help available

**return**
list_mode_adv: No help available

**get_value**() → AutoStep

```
# SCPI: [SOURce<HW>]:LIST:MODE
value: enums.AutoStep = driver.source.listPy.mode.get_value()
```

Sets the list mode. The instrument processes the list according to the selected mode and trigger source. See LIST:TRIG:SOUR AUTO, SING or EXT for the description of the trigger source settings.

**return**
mode: AUTO| STEP AUTO Each trigger event triggers a complete list cycle. STEP Each trigger event triggers only one step in the list processing cycle. The list is processed in ascending order.

**set_advanced**(*list_mode_adv: AutoManualMode*) → None

```
# SCPI: [SOURce<HW>]:LIST:MODE:ADVanced
driver.source.listPy.mode.set_advanced(list_mode_adv = enums.AutoManualMode.
↪AUTO)
```

No command help available

**param list_mode_adv**
No help available

**set_value**(*mode: AutoStep*) → None

```
# SCPI: [SOURce<HW>]:LIST:MODE
driver.source.listPy.mode.set_value(mode = enums.AutoStep.AUTO)
```

Sets the list mode. The instrument processes the list according to the selected mode and trigger source. See LIST:TRIG:SOUR AUTO, SING or EXT for the description of the trigger source settings.

**param mode**
AUTO| STEP AUTO Each trigger event triggers a complete list cycle. STEP Each trigger event triggers only one step in the list processing cycle. The list is processed in ascending order.

#### 6.18.15.7 Power

**SCPI Commands :**

```
[SOURce<HW>]:LIST:POWer:AMODe
[SOURce<HW>]:LIST:POWer:POINts
[SOURce<HW>]:LIST:POWer
```

**class PowerCls**

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

**get_amode**() → PowAttMode

```
# SCPI: [SOURce<HW>]:LIST:POWer:AMODe
value: enums.PowAttMode = driver.source.listPy.power.get_amode()
```

No command help available

**return**
amode: No help available

**get_points**() → int

```
# SCPI: [SOURce<HW>]:LIST:POWer:POINts
value: int = driver.source.listPy.power.get_points()
```

Queries the number (points) of level entries in the selected list.

**return**
points: integer Range: 0 to INT_MAX

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:LIST:POWer
value: List[float] = driver.source.listPy.power.get_value()
```

Enters the level values in the selected list. The number of level values must correspond to the number of frequency values. Existing data is overwritten.

**return**
power: Power#1{, Power#2, ...} | block data You can either enter the data as a list of numbers, or as binary block data. The list of numbers can be of any length, with the list entries separated by commas. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See also method RsSmab.FormatPy.data. Range: depends on the installed options , Unit: dBm

**set_amode**(*amode: PowAttMode*) → None

```
# SCPI: [SOURce<HW>]:LIST:POWer:AMODe
driver.source.listPy.power.set_amode(amode = enums.PowAttMode.AUTO)
```

No command help available

**param amode**
No help available

**set_value**(*power: List[float]*) → None

```
# SCPI: [SOURce<HW>]:LIST:POWer
driver.source.listPy.power.set_value(power = [1.1, 2.2, 3.3])
```

Enters the level values in the selected list. The number of level values must correspond to the number of frequency values. Existing data is overwritten.

**param power**
Power#1{, Power#2, ...} | block data You can either enter the data as a list of numbers, or as binary block data. The list of numbers can be of any length, with the list entries separated by commas. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See also method RsSmab.FormatPy.data. Range: depends on the installed options , Unit: dBm

#### 6.18.15.8 Trigger

**class TriggerCls**

Trigger commands group definition. 3 total commands, 2 Subgroups, 0 group commands

##### Cloning the Group

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

##### Subgroups

###### 6.18.15.8.1 Execute

##### SCPI Command :

```
[SOURce<HW>]:LIST:TRIGger:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:EXECute
driver.source.listPy.trigger.execute.set()
```

Starts the processing of a list in list mode.

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

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:EXECute
driver.source.listPy.trigger.execute.set_with_opc()
```

Starts the processing of a list in list mode.

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

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

#### 6.18.15.8.2 Source

**SCPI Commands :**

```
[SOURce<HW>]:LIST:TRIGger:SOURce:ADVanced
[SOURce<HW>]:LIST:TRIGger:SOURce
```

**class SourceCls**

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

**get_advanced**() → TrigSweepImmBusExt

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.source.listPy.trigger.source.get_
↪advanced()
```

No command help available

**return**
trig_point_adv: No help available

**get_value**() → TrigSweepSourNoHopExtAuto

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:SOURce
value: enums.TrigSweepSourNoHopExtAuto = driver.source.listPy.trigger.source.
↪get_value()
```

Selects the trigger source for processing lists. The designation of the parameters correspond to those in sweep mode. SCPI standard uses other designations for the parameters, which are also accepted by the instrument. The SCPI designation should be used if compatibility is an important consideration. For an overview, see the following table:

Table Header: Rohde & Schwarz parameter / SCPI parameter / Applies to the list mode parameters:

- AUTO / IMMediate / [:SOURce<hw>]:LIST:MODE AUTO
- SINGle / BUS / [:SOURce<hw>]:LIST:MODE AUTO or [:SOURce<hw>]:LIST:MODE STEP
- EXTernal / EXTernal / [:SOURce<hw>]:LIST:MODE AUTO or [:SOURce<hw>]:LIST:MODE STEP

**return**
source: AUTO| IMMediate| SINGle| BUS| EXTernal AUTO|IMMediate The trigger is free-running, i.e. the trigger condition is fulfilled continuously. The selected list is restarted as soon as it is finished. SINGle|BUS The list is triggered by the command [:SOURcehw]:LIST:TRIGger:EXECute. The list is executed once. EXTernal The list is triggered externally and executed once.

**set_advanced**(*trig_point_adv: TrigSweepImmBusExt*) → None

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:SOURce:ADVanced
driver.source.listPy.trigger.source.set_advanced(trig_point_adv = enums.
↪TrigSweepImmBusExt.BUS)
```

No command help available

**param trig_point_adv**
No help available

**set_value**(*source: TrigSweepSourNoHopExtAuto*) → None

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:SOURce
driver.source.listPy.trigger.source.set_value(source = enums.
↪TrigSweepSourNoHopExtAuto.AUTO)
```

Selects the trigger source for processing lists. The designation of the parameters correspond to those in sweep mode. SCPI standard uses other designations for the parameters, which are also accepted by the instrument. The SCPI designation should be used if compatibility is an important consideration. For an overview, see the following table:

Table Header: Rohde & Schwarz parameter / SCPI parameter / Applies to the list mode parameters:

- AUTO / IMMediate / [:SOURce<hw>]:LIST:MODE AUTO
- SINGle / BUS / [:SOURce<hw>]:LIST:MODE AUTO or [:SOURce<hw>]:LIST:MODE STEP
- EXTernal / EXTernal / [:SOURce<hw>]:LIST:MODE AUTO or [:SOURce<hw>]:LIST:MODE STEP

**param source**
AUTO| IMMediate| SINGle| BUS| EXTernal AUTO|IMMediate The trigger is free-running, i.e. the trigger condition is fulfilled continuously. The selected list is restarted as soon as it is finished. SINGle|BUS The list is triggered by the command [:SOURcehw]:LIST:TRIGger:EXECute. The list is executed once. EXTernal The list is triggered externally and executed once.

### 6.18.16 Mbeacon

#### SCPI Command :

```
[SOURce<HW>]:MBEacon:STATe
```

**class MbeaconCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:MBEacon:STATe
value: bool = driver.source.mbeacon.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:MBEacon:STATe
driver.source.mbeacon.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.18.17 Modulation

**class ModulationCls**

Modulation 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.source.modulation.clone()
```

#### Subgroups

##### 6.18.17.1 All

#### SCPI Command :

```
[SOURce<HW>]:MODulation:[ALL]:[STATe]
```

**class AllCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:MODulation:[ALL]:[STATe]
value: bool = driver.source.modulation.all.get_state()
```

Activates all modulations that were active before the last switching off.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:MODulation:[ALL]:[STATe]
driver.source.modulation.all.set_state(state = False)
```

Activates all modulations that were active before the last switching off.

**param state**
1| ON| 0| OFF

### 6.18.18 Noise

#### SCPI Command :

```
[SOURce<HW>]:NOISe:DISTribution
```

**class NoiseCls**

Noise commands group definition. 5 total commands, 2 Subgroups, 1 group commands

**get_distribution**() → NoisDistrib

```
# SCPI: [SOURce<HW>]:NOISe:DISTribution
value: enums.NoisDistrib = driver.source.noise.get_distribution()
```

Sets the distribution of the noise power density.

**return**
distribution: GAUSs| EQUal

**set_distribution**(*distribution: NoisDistrib*) → None

```
# SCPI: [SOURce<HW>]:NOISe:DISTribution
driver.source.noise.set_distribution(distribution = enums.NoisDistrib.EQUal)
```

Sets the distribution of the noise power density.

**param distribution**
GAUSs| EQUal

#### Cloning the Group

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

#### Subgroups

##### 6.18.18.1 Bandwidth

#### SCPI Commands :

```
[SOURce<HW>]:NOISe:BWIDth:STATe
[SOURce<HW>]:NOISe:BANDwidth
```

**class BandwidthCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:NOISe:BWIDth:STATe
value: bool = driver.source.noise.bandwidth.get_state()
```

Activates noise bandwidth limitation.

**return**
state: 1| ON| 0| OFF

**get_value**() → float

```
# SCPI: [SOURce<HW>]:NOISe:BANDwidth
value: float = driver.source.noise.bandwidth.get_value()
```

Sets the noise level in the system bandwidth when bandwidth limitation is enabled.

**return**
bwidth: float Range: 100E3 to 10E6

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:NOISe:BWIDth:STATe
driver.source.noise.bandwidth.set_state(state = False)
```

Activates noise bandwidth limitation.

**param state**
1| ON| 0| OFF

**set_value**(*bwidth: float*) → None

```
# SCPI: [SOURce<HW>]:NOISe:BANDwidth
driver.source.noise.bandwidth.set_value(bwidth = 1.0)
```

Sets the noise level in the system bandwidth when bandwidth limitation is enabled.

**param bwidth**
float Range: 100E3 to 10E6

#### 6.18.18.2 Level

**SCPI Commands :**

```
[SOURce<HW>]:NOISe:LEVel:RELative
[SOURce<HW>]:NOISe:LEVel:[ABSolute]
```

**class LevelCls**

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

**get_absolute**() → float

```
# SCPI: [SOURce<HW>]:NOISe:LEVel:[ABSolute]
value: float = driver.source.noise.level.get_absolute()
```

Queries the level of the noise signal in the system bandwidth within the enabled bandwidth limitation.

**return**
absolute: float Noise level within the bandwidth limitation

**get_relative**() → float

```
# SCPI: [SOURce<HW>]:NOISe:LEVel:RELative
value: float = driver.source.noise.level.get_relative()
```

Queries the level of the noise signal per Hz in the total bandwidth.

**return**
relative: float Range: -149.18 to -52.67

### 6.18.19 Path

#### SCPI Command :

```
[SOURce]:PATH:COUNt
```

**class PathCls**

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

**get_count**() → int

```
# SCPI: [SOURce]:PATH:COUNt
value: int = driver.source.path.get_count()
```

No command help available

**return**
count: No help available

### 6.18.20 Pgenerator

#### SCPI Command :

```
[SOURce<HW>]:PGENerator:STATe
```

**class PgeneratorCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:PGENerator:STATe
value: bool = driver.source.pgenerator.get_state()
```

Enables the output of the video/sync signal. If the pulse generator is the current modulation source, activating the pulse modulation automatically activates the signal output and the pulse generator.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:PGENerator:STATe
driver.source.pgenerator.set_state(state = False)
```

Enables the output of the video/sync signal. If the pulse generator is the current modulation source, activating the pulse modulation automatically activates the signal output and the pulse generator.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.20.1 Output

### SCPI Commands :

```
[SOURce<HW>]:PGENerator:OUTPut:POLarity
[SOURce<HW>]:PGENerator:OUTPut:[STATe]
```

**class OutputCls**

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

**get_polarity**() → NormalInverted

```
# SCPI: [SOURce<HW>]:PGENerator:OUTPut:POLarity
value: enums.NormalInverted = driver.source.pgenerator.output.get_polarity()
```

Sets the polarity of the pulse output signal.

**return**
polarity: NORMal| INVerted NORMal Outputs the pulse signal during the pulse width, that means during the high state. INVerted Inverts the pulse output signal polarity. The pulse output signal is suppressed during the pulse width, but provided during the low state.

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:PGENerator:OUTPut:[STATe]
value: bool = driver.source.pgenerator.output.get_state()
```

Activates the output of the pulse modulation signal.

**return**
state: 1| ON| 0| OFF

**set_polarity**(*polarity: NormalInverted*) → None

```
# SCPI: [SOURce<HW>]:PGENerator:OUTPut:POLarity
driver.source.pgenerator.output.set_polarity(polarity = enums.NormalInverted.
↪INVerted)
```

Sets the polarity of the pulse output signal.

**param polarity**
NORMal| INVerted NORMal Outputs the pulse signal during the pulse width, that means during the high state. INVerted Inverts the pulse output signal polarity. The pulse output signal is suppressed during the pulse width, but provided during the low state.

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:PGENerator:OUTPut:[STATe]
driver.source.pgenerator.output.set_state(state = False)
```

Activates the output of the pulse modulation signal.

**param state**
1| ON| 0| OFF

### 6.18.21 Phase

#### SCPI Command :

```
[SOURce<HW>]:PHASe
```

**class PhaseCls**

Phase commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_value**() → float

```
# SCPI: [SOURce<HW>]:PHASe
value: float = driver.source.phase.get_value()
```

Sets the phase variation relative to the current phase.

**return**
phase: float Range: -36000 to 36000 , Unit: DEG

**set_value**(*phase: float*) → None

```
# SCPI: [SOURce<HW>]:PHASe
driver.source.phase.set_value(phase = 1.0)
```

Sets the phase variation relative to the current phase.

**param phase**
float Range: -36000 to 36000 , Unit: DEG

#### Cloning the Group

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

#### Subgroups

##### 6.18.21.1 Reference

#### SCPI Command :

```
[SOURce<HW>]:PHASe:REFerence
```

**class ReferenceCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:PHASe:REFerence
driver.source.phase.reference.set()
```

Assigns the value set with command [:SOURce<hw>]:PHASe as the reference phase.

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

```
# SCPI: [SOURce<HW>]:PHASe:REFerence
driver.source.phase.reference.set_with_opc()
```

Assigns the value set with command [:SOURce<hw>]:PHASe as the reference phase.

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

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

### 6.18.22 Pm<GeneratorIx>

#### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.pm.repcap_generatorIx_get()
driver.source.pm.repcap_generatorIx_set(repcap.GeneratorIx.Nr1)
```

#### SCPI Commands :

```
[SOURce<HW>]:PM:MODE
[SOURce<HW>]:PM:RATio
[SOURce<HW>]:PM:SENSitivity
```

**class PmCls**

Pm commands group definition. 8 total commands, 3 Subgroups, 3 group commands Repeated Capability: GeneratorIx, default value after init: GeneratorIx.Nr1

**get_mode**() → PmMode

```
# SCPI: [SOURce<HW>]:PM:MODE
value: enums.PmMode = driver.source.pm.get_mode()
```

Selects the mode for the phase modulation.

**return**
mode: HBANdwidth| HDEViation| LNOise HBANdwidth Sets the maximum available bandwidth. HDEViation Sets the maximum range for FiM deviation. LNOise Selects a phase modulation mode with phase noise and spurious characteristics close to CW mode.

**get_ratio**() → float

```
# SCPI: [SOURce<HW>]:PM:RATio
value: float = driver.source.pm.get_ratio()
```

Sets the deviation ratio (path2 to path1) in percent.

**return**
ratio: float Range: 0 to 100

**get_sensitivity**() → float

```
# SCPI: [SOURce<HW>]:PM:SENSitivity
value: float = driver.source.pm.get_sensitivity()
```

Queries the sensitivity of the externally applied signal for phase modulation. The returned value reports the sensitivity in RAD/V. It is assigned to the voltage value for full modulation of the input.

**return**
sensitivity: float

**set_mode**(*mode: PmMode*) → None

```
# SCPI: [SOURce<HW>]:PM:MODE
driver.source.pm.set_mode(mode = enums.PmMode.HBANdwidth)
```

Selects the mode for the phase modulation.

**param mode**
HBANdwidth| HDEViation| LNOise HBANdwidth Sets the maximum available bandwidth. HDEViation Sets the maximum range for FiM deviation. LNOise Selects a phase modulation mode with phase noise and spurious characteristics close to CW mode.

**set_ratio**(*ratio: float*) → None

```
# SCPI: [SOURce<HW>]:PM:RATio
driver.source.pm.set_ratio(ratio = 1.0)
```

Sets the deviation ratio (path2 to path1) in percent.

**param ratio**
float Range: 0 to 100

### Cloning the Group

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

### Subgroups

#### 6.18.22.1 Deviation

### SCPI Commands :

```
[SOURce<HW>]:PM:DEViation:MODE
[SOURce<HW>]:PM:DEViation:SUM
[SOURce]:PM<CH>:[DEViation]
```

**class DeviationCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → float

```
# SCPI: [SOURce]:PM<CH>:[DEViation]
value: float = driver.source.pm.deviation.get(generatorIx = repcap.GeneratorIx.
↪Default)
```

Sets the modulation deviation of the phase modulation in RAD.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Pm')

**return**
deviation: float The maximal deviation depends on the RF frequency and the selected modulation mode (see data sheet) . Range: 0 to max, Unit: RAD

**get_mode**() → ModulationDevMode

```
# SCPI: [SOURce<HW>]:PM:DEViation:MODE
value: enums.ModulationDevMode = driver.source.pm.deviation.get_mode()
```

Selects the coupling mode. The coupling mode parameter also determines the mode for fixing the total deviation.

**return**
pm_dev_mode: UNCoupled| TOTal| RATio UNCoupled Does not couple the LF signals. The deviation values of both paths are independent. TOTal Couples the deviation of both paths. RATio Couples the deviation ratio of both paths

**get_sum**() → float

```
# SCPI: [SOURce<HW>]:PM:DEViation:SUM
value: float = driver.source.pm.deviation.get_sum()
```

Sets the total deviation of the LF signal when using combined signal sources in phase modulation.

**return**
pm_dev_sum: float Range: 0 to max

**set**(*deviation: float*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce]:PM<CH>:[DEViation]
driver.source.pm.deviation.set(deviation = 1.0, generatorIx = repcap.
↪GeneratorIx.Default)
```

Sets the modulation deviation of the phase modulation in RAD.

**param deviation**
float The maximal deviation depends on the RF frequency and the selected modulation mode (see data sheet) . Range: 0 to max, Unit: RAD

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Pm')

**set_mode**(*pm_dev_mode: ModulationDevMode*) → None

```
# SCPI: [SOURce<HW>]:PM:DEViation:MODE
driver.source.pm.deviation.set_mode(pm_dev_mode = enums.ModulationDevMode.RATio)
```

Selects the coupling mode. The coupling mode parameter also determines the mode for fixing the total deviation.

**param pm_dev_mode**
UNCoupled| TOTal| RATio UNCoupled Does not couple the LF signals. The deviation values of both paths are independent. TOTal Couples the deviation of both paths. RATio Couples the deviation ratio of both paths

**set_sum**(*pm_dev_sum: float*) → None

```
# SCPI: [SOURce<HW>]:PM:DEViation:SUM
driver.source.pm.deviation.set_sum(pm_dev_sum = 1.0)
```

Sets the total deviation of the LF signal when using combined signal sources in phase modulation.

**param pm_dev_sum**
float Range: 0 to max

#### 6.18.22.2 Source

##### SCPI Command :

```
[SOURce<HW>]:PM<CH>:SOURce
```

**class SourceCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → FmSour

```
# SCPI: [SOURce<HW>]:PM<CH>:SOURce
value: enums.FmSour = driver.source.pm.source.get(generatorIx = repcap.
↪GeneratorIx.Default)
```

Selects the modulation source for phase modulation signal.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Pm')

**return**
source: LF1| LF2| NOISe| EXT1| EXT2| INTernal| EXTernal LF1|LF2 Uses an internally generated LF signal. EXT1|EXT2 Uses an externally supplied LF signal. NOISe Uses the internally generated noise signal. INTernal Uses the internally generated signal of LF1. EXTernal Uses an external LF signal (EXT1) .

**set**(*source: FmSour*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:PM<CH>:SOURce
driver.source.pm.source.set(source = enums.FmSour.EXT1, generatorIx = repcap.
↪GeneratorIx.Default)
```

Selects the modulation source for phase modulation signal.

**param source**
LF1| LF2| NOISe| EXT1| EXT2| INTernal| EXTernal LF1|LF2 Uses an internally generated LF signal. EXT1|EXT2 Uses an externally supplied LF signal. NOISe Uses the internally generated noise signal. INTernal Uses the internally generated signal of LF1. EXTernal Uses an external LF signal (EXT1) .

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Pm')

#### 6.18.22.3 State

#### SCPI Command :

```
[SOURce<HW>]:PM<CH>:STATe
```

**class StateCls**

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

**get**(*generatorIx=GeneratorIx.Default*) → bool

```
# SCPI: [SOURce<HW>]:PM<CH>:STATe
value: bool = driver.source.pm.state.get(generatorIx = repcap.GeneratorIx.
↪Default)
```

Activates phase modulation. Activation of phase modulation deactivates frequency modulation.

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Pm')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *generatorIx=GeneratorIx.Default*) → None

```
# SCPI: [SOURce<HW>]:PM<CH>:STATe
driver.source.pm.state.set(state = False, generatorIx = repcap.GeneratorIx.
↪Default)
```

Activates phase modulation. Activation of phase modulation deactivates frequency modulation.

**param state**
1| ON| 0| OFF

**param generatorIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Pm')

### 6.18.23 Power

#### SCPI Commands :

```
[SOURce<HW>]:POWer:LBEHaviour
[SOURce<HW>]:POWer:LMODe
[SOURce<HW>]:POWer:MANual
[SOURce<HW>]:POWer:MODE
[SOURce<HW>]:POWer:POWer
[SOURce<HW>]:POWer:STARt
[SOURce<HW>]:POWer:STOP
[SOURce]:POWer:WIGNore
```

**class PowerCls**

Power commands group definition. 42 total commands, 8 Subgroups, 8 group commands

**get_lbehaviour**() → PowLevBehaviour

```
# SCPI: [SOURce<HW>]:POWer:LBEHaviour
value: enums.PowLevBehaviour = driver.source.power.get_lbehaviour()
```

No command help available

**return**
behaviour: AUTO| UNINterrupted| MONotone| CVSWr| HDUN UNINterrupted|MONotone Uninterrupted level settings and strictly monotone modes. CVSWr Constant VSWR HDUN High dynamic uninterrupted level settings.

**get_lmode**() → PowLevMode

```
# SCPI: [SOURce<HW>]:POWer:LMODe
value: enums.PowLevMode = driver.source.power.get_lmode()
```

No command help available

**return**
lev_mode: No help available

**get_manual**() → float

```
# SCPI: [SOURce<HW>]:POWer:MANual
value: float = driver.source.power.get_manual()
```

Sets the level for the subsequent sweep step if [:SOURce<hw>]:SWEep:POWer:MODE. Use a separate command for each sweep step.

**return**
manual: float You can select any level within the setting range, where: STARt is set with [:SOURcehw]:POWer:STARt STOP is set with [:SOURcehw]:POWer:STOP OFFSet is set with [:SOURcehw]:POWer[:LEVel][:IMMediate]:OFFSet Range: (STARt + OFFSet) to (STOP + OFFSet) , Unit: dBm

**get_mode**() → LfFreqMode

```
# SCPI: [SOURce<HW>]:POWer:MODE
value: enums.LfFreqMode = driver.source.power.get_mode()
```

Selects the operating mode of the instrument to set the output level.

**return**
mode: CW| FIXed| SWEep CW|FIXed Operates at a constant level. CW and FIXed are synonyms. To set the output level value, use the command [:SOURcehw]:POWer[:LEVel][:IMMediate][:AMPLitude]. SWEep Sets sweep mode. Set the range and current level with the commands: [:SOURcehw]:POWer:STARt and [:SOURcehw]:POWer:STOP, [:SOURcehw]:POWer:MANual.

**get_power**() → float

```
# SCPI: [SOURce<HW>]:POWer:POWer
value: float = driver.source.power.get_power()
```

Sets the level at the RF output connector. This value does not consider a specified offset. The command [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude] sets the level of the 'Level' display, that means the level containing offset. See 'RF frequency and level display with a downstream instrument'.

**return**
power: float Level at the RF output, without level offset Range: See data sheet , Unit: dBm

**get_start**() → float

```
# SCPI: [SOURce<HW>]:POWer:STARt
value: float = driver.source.power.get_start()
```

Sets the RF start/stop level in sweep mode.

**return**
start: No help available

**get_stop**() → float

```
# SCPI: [SOURce<HW>]:POWer:STOP
value: float = driver.source.power.get_stop()
```

Sets the RF start/stop level in sweep mode.

**return**
stop: float Sets the setting range calculated as follows: (Level_min + OFFSet) to (Level_max + OFFSet) Where the values are set with the commands: [:SOURcehw]:POWer[:LEVel][:IMMediate]:OFFSet [:SOURcehw]:POWer:STARt [:SOURcehw]:POWer:STOP Range: Minimum level to maximum level , Unit: dBm

**get_wignore**() → bool

```
# SCPI: [SOURce]:POWer:WIGNore
value: bool = driver.source.power.get_wignore()
```

Ignores level range warnings.

**return**
state: 1| ON| 0| OFF

**set_lbehaviour**(*behaviour: PowLevBehaviour*) → None

```
# SCPI: [SOURce<HW>]:POWer:LBEHaviour
driver.source.power.set_lbehaviour(behaviour = enums.PowLevBehaviour.AUTO)
```

No command help available

**param behaviour**
AUTO| UNINterrupted| MONotone| CVSWr| HDUN UNINterrupted|MONotone Uninterrupted level settings and strictly monotone modes. CVSWr Constant VSWR HDUN High dynamic uninterrupted level settings.

**set_lmode**(*lev_mode: PowLevMode*) → None

```
# SCPI: [SOURce<HW>]:POWer:LMODe
driver.source.power.set_lmode(lev_mode = enums.PowLevMode.LOWDistortion)
```

No command help available

**param lev_mode**
No help available

**set_manual**(*manual: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:MANual
driver.source.power.set_manual(manual = 1.0)
```

Sets the level for the subsequent sweep step if [:SOURce<hw>]:SWEep:POWer:MODE. Use a separate command for each sweep step.

**param manual**
float You can select any level within the setting range, where: STARt is set with [:SOURcehw]:POWer:STARt STOP is set with [:SOURcehw]:POWer:STOP OFFSet is set with [:SOURcehw]:POWer[:LEVel][:IMMediate]:OFFSet Range: (STARt + OFFSet) to (STOP + OFFSet) , Unit: dBm

**set_mode**(*mode: LfFreqMode*) → None

```
# SCPI: [SOURce<HW>]:POWer:MODE
driver.source.power.set_mode(mode = enums.LfFreqMode.CW)
```

Selects the operating mode of the instrument to set the output level.

**param mode**
CW| FIXed| SWEep CW|FIXed Operates at a constant level. CW and FIXed are synonyms. To set the output level value, use the command [:SOURcehw]:POWer[:LEVel][:IMMediate][:AMPLitude]. SWEep

Sets sweep mode. Set the range and current level with the commands: [:SOURcehw]:POWer:STARt and [:SOURcehw]:POWer:STOP, [:SOURcehw]:POWer:MANual.

**set_power**(*power: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:POWer
driver.source.power.set_power(power = 1.0)
```

Sets the level at the RF output connector. This value does not consider a specified offset. The command [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude] sets the level of the 'Level' display, that means the level containing offset. See 'RF frequency and level display with a downstream instrument'.

**param power**
float Level at the RF output, without level offset Range: See data sheet , Unit: dBm

**set_start**(*start: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:STARt
driver.source.power.set_start(start = 1.0)
```

Sets the RF start/stop level in sweep mode.

**param start**
float Sets the setting range calculated as follows: (Level_min + OFFSet) to (Level_max + OFFSet) Where the values are set with the commands: [:SOURcehw]:POWer[:LEVel][:IMMediate]:OFFSet [:SOURcehw]:POWer:STARt [:SOURcehw]:POWer:STOP Range: Minimum level to maximum level , Unit: dBm

**set_stop**(*stop: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:STOP
driver.source.power.set_stop(stop = 1.0)
```

Sets the RF start/stop level in sweep mode.

**param stop**
float Sets the setting range calculated as follows: (Level_min + OFFSet) to (Level_max + OFFSet) Where the values are set with the commands: [:SOURcehw]:POWer[:LEVel][:IMMediate]:OFFSet [:SOURcehw]:POWer:STARt [:SOURcehw]:POWer:STOP Range: Minimum level to maximum level , Unit: dBm

**set_wignore**(*state: bool*) → None

```
# SCPI: [SOURce]:POWer:WIGNore
driver.source.power.set_wignore(state = False)
```

Ignores level range warnings.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.23.1 Alc

### SCPI Commands :

```
[SOURce<HW>]:POWer:ALC:DSENsitivity
[SOURce<HW>]:POWer:ALC:MODE
[SOURce<HW>]:POWer:ALC:OMODe
[SOURce<HW>]:POWer:ALC:SEARch
[SOURce<HW>]:POWer:ALC:[STATe]
```

**class AlcCls**

Alc commands group definition. 8 total commands, 2 Subgroups, 5 group commands

**get_dsensitivity**() → PowAlcDetSensitivity

```
# SCPI: [SOURce<HW>]:POWer:ALC:DSENsitivity
value: enums.PowAlcDetSensitivity = driver.source.power.alc.get_dsensitivity()
```

Sets the sensitivity of the ALC detector.

**return**
sensitivity: AUTO| FIXed AUTO Selects the optimum sensitivity automatically. FIXed Fixes the internal level detector.

**get_mode**() → AlcOnOffAuto

```
# SCPI: [SOURce<HW>]:POWer:ALC:MODE
value: enums.AlcOnOffAuto = driver.source.power.alc.get_mode()
```

Queries the currently set ALC mode. See [:SOURce<hw>]:POWer:ALC[:STATe].

**return**
pow_alc_mode: 0| AUTO| 1| PRESet| OFFTable| ON| OFF| ONSample| ONTable

**get_omode**() → AlcOffMode

```
# SCPI: [SOURce<HW>]:POWer:ALC:OMODe
value: enums.AlcOffMode = driver.source.power.alc.get_omode()
```

No command help available

**return**
off_mode: No help available

**get_search**() → bool

```
# SCPI: [SOURce<HW>]:POWer:ALC:SEARch
value: bool = driver.source.power.alc.get_search()
```

No command help available

**return**
search: No help available

**get_state**() → PowAlcStateWithExtAlc

```
# SCPI: [SOURce<HW>]:POWer:ALC:[STATe]
value: enums.PowAlcStateWithExtAlc = driver.source.power.alc.get_state()
```

No command help available

**return**
state: 0| OFF| AUTO| 1| ON| ONTable| PRESet| OFFTable | EALC AUTO Adjusts the output level to the operating conditions automatically. 1|ON Activates internal level control permanently. OFFTable Controls the level using attenuation values of the internal ALC table. 0|OFF Provided only for backward compatibility with other Rohde & Schwarz signal generators. The R&S SMA100B accepts these values and maps them automatically as follows: 0|OFF = OFFTable ONTable Starts with the attenuation setting from the table and continues with automatic level control. EALC Activates external ALC mode.

**set_dsensitivity**(*sensitivity: PowAlcDetSensitivity*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:DSENsitivity
driver.source.power.alc.set_dsensitivity(sensitivity = enums.
↪PowAlcDetSensitivity.AUTO)
```

Sets the sensitivity of the ALC detector.

**param sensitivity**
AUTO| FIXed AUTO Selects the optimum sensitivity automatically. FIXed Fixes the internal level detector.

**set_omode**(*off_mode: AlcOffMode*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:OMODe
driver.source.power.alc.set_omode(off_mode = enums.AlcOffMode.SHOLd)
```

No command help available

**param off_mode**
No help available

**set_search**(*search: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:SEARch
driver.source.power.alc.set_search(search = False)
```

No command help available

**param search**
No help available

**set_state**(*state: PowAlcStateWithExtAlc*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:[STATe]
driver.source.power.alc.set_state(state = enums.PowAlcStateWithExtAlc._0)
```

No command help available

**param state**
0| OFF| AUTO| 1| ON| ONTable| PRESet| OFFTable | EALC AUTO Adjusts the output level to the operating conditions automatically. 1|ON Activates internal level control permanently. OFFTable Controls the level using attenuation values of the internal ALC table. 0|OFF Provided only for backward compatibility with other Rohde & Schwarz signal generators. The R&S SMA100B accepts these values and maps them automatically as follows: 0|OFF = OFFTable ONTable Starts with the attenuation setting from the table and continues with automatic level control. EALC Activates external ALC mode.

### Cloning the Group

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

### Subgroups

#### 6.18.23.1.1 Edetector

### SCPI Commands :

```
[SOURce<HW>]:POWer:ALC:EDETector:FACTor
[SOURce<HW>]:POWer:ALC:EDETector:LEVel
```

**class EdetectorCls**

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

**get_factor**() → float

```
# SCPI: [SOURce<HW>]:POWer:ALC:EDETector:FACTor
value: float = driver.source.power.alc.edetector.get_factor()
```

Sets the attenuation value of the RF coupler.

**return**
detector_fact: float Range: -200 to 200

**get_level**() → float

```
# SCPI: [SOURce<HW>]:POWer:ALC:EDETector:LEVel
value: float = driver.source.power.alc.edetector.get_level()
```

Sets the maximum power level at the RF output required for compensating the external ALC coupler and cable losses.

**return**
req_gen_lev: float Range: -145 to 40

#### 6.18.23.1.2 Sonce

**SCPI Command :**

```
[SOURce<HW>]:POWer:ALC:SONCe
```

**class SonceCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:SONCe
driver.source.power.alc.sonce.set()
```

Activates level control for correction purposes temporarily.

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

```
# SCPI: [SOURce<HW>]:POWer:ALC:SONCe
driver.source.power.alc.sonce.set_with_opc()
```

Activates level control for correction purposes temporarily.

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

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

### 6.18.23.2 Attenuation

**SCPI Commands :**

```
[SOURce<HW>]:POWer:ATTenuation:MAXLevel
[SOURce<HW>]:POWer:ATTenuation:PATTenuator
[SOURce<HW>]:POWer:ATTenuation:STAGe
```

**class AttenuationCls**

Attenuation commands group definition. 4 total commands, 1 Subgroups, 3 group commands

**get_max_level**() → float

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:MAXLevel
value: float = driver.source.power.attenuation.get_max_level()
```

No command help available

**return**
level: No help available

**get_pattenuator**() → PowAttStepArt

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:PATTenuator
value: enums.PowAttStepArt = driver.source.power.attenuation.get_pattenuator()
```

Selects the type of step attenuator used below 20 GHz.

**return**
step_att_sel: MECHanical| ELECtronic MECHanical Uses the mechanical step attenuator at all frequencies. ELECtronic Uses the electronic step attenuator up to 20 GHz.

**get_stage**() → float

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:STAGe
value: float = driver.source.power.attenuation.get_stage()
```

No command help available

**return**
stage: No help available

**set_max_level**(*level: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:MAXLevel
driver.source.power.attenuation.set_max_level(level = 1.0)
```

No command help available

**param level**
No help available

**set_pattenuator**(*step_att_sel: PowAttStepArt*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:PATTenuator
driver.source.power.attenuation.set_pattenuator(step_att_sel = enums.
↪PowAttStepArt.ELECtronic)
```

Selects the type of step attenuator used below 20 GHz.

**param step_att_sel**
MECHanical| ELECtronic MECHanical Uses the mechanical step attenuator at all frequencies. ELECtronic Uses the electronic step attenuator up to 20 GHz.

### Cloning the Group

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

### Subgroups

#### 6.18.23.2.1 RfOff

### SCPI Command :

```
[SOURce<HW>]:POWer:ATTenuation:RFOFf:MODE
```

**class RfOffCls**

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

**get_mode**() → PowAttRfOffMode

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:RFOFf:MODE
value: enums.PowAttRfOffMode = driver.source.power.attenuation.rfOff.get_mode()
```

Selects the state the attenuator is to assume if the RF signal is switched off.

**return**
mode: UNCHanged| FATTenuation FATTenuation The step attenuator switches to maximum attenuation UNCHanged Retains the current setting and keeps the output impedance constant during RF off.

**set_mode**(*mode: PowAttRfOffMode*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:RFOFf:MODE
driver.source.power.attenuation.rfOff.set_mode(mode = enums.PowAttRfOffMode.
↪FATTenuation)
```

Selects the state the attenuator is to assume if the RF signal is switched off.

**param mode**
UNCHanged| FATTenuation FATTenuation The step attenuator switches to maximum attenuation UNCHanged Retains the current setting and keeps the output impedance constant during RF off.

#### 6.18.23.3 Emf

**SCPI Command :**

```
[SOURce<HW>]:POWer:EMF:STATe
```

**class EmfCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:POWer:EMF:STATe
value: bool = driver.source.power.emf.get_state()
```

Displays the signal level as voltage of the EMF. The displayed value represents the voltage over a 50 Ohm load.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:EMF:STATe
driver.source.power.emf.set_state(state = False)
```

Displays the signal level as voltage of the EMF. The displayed value represents the voltage over a 50 Ohm load.

**param state**
1| ON| 0| OFF

#### 6.18.23.4 Level

**class LevelCls**

Level 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.source.power.level.clone()
```

### Subgroups

#### 6.18.23.4.1 Immediate

### SCPI Commands :

```
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:OFFSet
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:REFLevel
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:[AMPLitude]
```

**class ImmediateCls**

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

**get_amplitude**() → float

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:[AMPLitude]
value: float = driver.source.power.level.immediate.get_amplitude()
```

Sets the RF level applied to the DUT. To activate the RF output use command method RsSmab.Output.State.value ('RF On'/'RF Off') .

INTRO_CMD_HELP: The following applies POWer = RF output level + OFFSet, where:

- POWer is the values set with [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude]
- RF output level is set with [:SOURce<hw>]:POWer:POWer
- OFFSet is set with [:SOURce<hw>]:POWer[:LEVel][:IMMediate]:OFFSet

**return**

amplitude: float The following settings influence the value range: OFFSet set with the command [:SOURcehw]:POWer[:LEVel][:IMMediate]:OFFSet Numerical value Sets the level UP|DOWN Varies the level step by step. The level is increased or decreased by the value set with the command [:SOURcehw]:POWer:STEP[:INCRement]. Range: (Level_min + OFFSet) to (Level_max + OFFStet) , Unit: dBm

**get_offset**() → float

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:OFFSet
value: float = driver.source.power.level.immediate.get_offset()
```

Sets the level offset of a downstream instrument. The level at the RF output is not changed. To query the resulting level, as it is at the output of the downstream instrument, use the command [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude]. See 'RF frequency and level display with a downstream instrument'. Note: The level offset also affects the RF level sweep.

**return**
offset: float Range: -100 to 100 , Unit: dB Level offset is always expreced in dB; linear units (V, W, etc.) are not supported

**get_recall**() → InclExcl

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
value: enums.InclExcl = driver.source.power.level.immediate.get_recall()
```

Determines whether the current level is retained or if the stored level setting is adopted when an instrument configuration is loaded.

**return**
rcl: INCLude| EXCLude INCLude Takes the current level when an instrument configuration is loaded. EXCLude Retains the current level when an instrument configuration is loaded.

**get_ref_level**() → float

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:REFLevel
value: float = driver.source.power.level.immediate.get_ref_level()
```

Queries the reference level of the user correction. The reference level is the sum of the amplitude and the level offset, set with the commands [:SOURce<hw>]:POWer:POWer [:SOURce<hw>]:POWer[:LEVel][:IMMediate]:OFFSet.

**return**
reference_level: float Range: -245 to 120

**set_amplitude**(*amplitude: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:[AMPLitude]
driver.source.power.level.immediate.set_amplitude(amplitude = 1.0)
```

Sets the RF level applied to the DUT. To activate the RF output use command method RsSmab.Output.State.value ('RF On'/'RF Off') .

INTRO_CMD_HELP: The following applies POWer = RF output level + OFFSet, where:

- POWer is the values set with [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude]
- RF output level is set with [:SOURce<hw>]:POWer:POWer
- OFFSet is set with [:SOURce<hw>]:POWer[:LEVel][:IMMediate]:OFFSet

**param amplitude**
float The following settings influence the value range: OFFSet set with the command [:SOURcehw]:POWer[:LEVel][:IMMediate]:OFFSet Numerical value Sets the level UP|DOWN Varies the level step by step. The level is increased or decreased by the value set with the command [:SOURcehw]:POWer:STEP[:INCRement]. Range: (Level_min + OFFSet) to (Level_max + OFFStet) , Unit: dBm

**set_offset**(*offset: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:OFFSet
driver.source.power.level.immediate.set_offset(offset = 1.0)
```

Sets the level offset of a downstream instrument. The level at the RF output is not changed. To query the resulting level, as it is at the output of the downstream instrument, use the command [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude]. See 'RF frequency and level display with a downstream instrument'. Note: The level offset also affects the RF level sweep.

**param offset**
float Range: -100 to 100 , Unit: dB Level offset is always expreced in dB; linear units (V, W, etc.) are not supported

**set_recall**(*rcl: InclExcl*) → None

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
driver.source.power.level.immediate.set_recall(rcl = enums.InclExcl.EXCLude)
```

Determines whether the current level is retained or if the stored level setting is adopted when an instrument configuration is loaded.

**param rcl**
INCLude| EXCLude INCLude Takes the current level when an instrument configuration is loaded. EXCLude Retains the current level when an instrument configuration is loaded.

**set_ref_level**(*reference_level: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:REFLevel
driver.source.power.level.immediate.set_ref_level(reference_level = 1.0)
```

Queries the reference level of the user correction. The reference level is the sum of the amplitude and the level offset, set with the commands [:SOURce<hw>]:POWer:POWer [:SOURce<hw>]:POWer[:LEVel][:IMMediate]:OFFSet.

**param reference_level**
float Range: -245 to 120

#### 6.18.23.5 Limit

**SCPI Command :**

```
[SOURce<HW>]:POWer:LIMit:[AMPLitude]
```

**class LimitCls**

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

**get_amplitude**() → float

```
# SCPI: [SOURce<HW>]:POWer:LIMit:[AMPLitude]
value: float = driver.source.power.limit.get_amplitude()
```

Limits the maximum RF output level in CW and sweep mode. It does not influence the 'Level' display or the response to the query [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude].

**return**
amplitude: float Range: depends on the installed options

**set_amplitude**(*amplitude: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:LIMit:[AMPLitude]
driver.source.power.limit.set_amplitude(amplitude = 1.0)
```

Limits the maximum RF output level in CW and sweep mode. It does not influence the 'Level' display or the response to the query [:SOURce<hw>]:POWer[:LEVel][:IMMediate][:AMPLitude].

**param amplitude**
float Range: depends on the installed options

#### 6.18.23.6 Range

**SCPI Commands :**

```
[SOURce<HW>]:POWer:RANGe:LOWer
[SOURce<HW>]:POWer:RANGe:MAX
[SOURce<HW>]:POWer:RANGe:MIN
[SOURce<HW>]:POWer:RANGe:UPPer
```

**class RangeCls**

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

**get_lower**() → float

```
# SCPI: [SOURce<HW>]:POWer:RANGe:LOWer
value: float = driver.source.power.range.get_lower()
```

Queries the current interruption-free range of the level.

**return**
lower: float Unit: dBm

**get_max**() → float

```
# SCPI: [SOURce<HW>]:POWer:RANGe:MAX
value: float = driver.source.power.range.get_max()
```

Queries the current power range of the level sweep.

**return**
pow_range_max: float Range: depends on settings , Unit: dBm

**get_min**() → float

```
# SCPI: [SOURce<HW>]:POWer:RANGe:MIN
value: float = driver.source.power.range.get_min()
```

Queries the current power range of the level sweep.

**return**
pow_range_min: float Range: depends on settings , Unit: dBm

**get_upper**() → float

```
# SCPI: [SOURce<HW>]:POWer:RANGe:UPPer
value: float = driver.source.power.range.get_upper()
```

Queries the current interruption-free range of the level.

**return**
upper: float Unit: dBm

#### 6.18.23.7 Spc

##### SCPI Commands :

```
[SOURce<HW>]:POWer:SPC:CRANge
[SOURce<HW>]:POWer:SPC:DELay
[SOURce<HW>]:POWer:SPC:MODE
[SOURce<HW>]:POWer:SPC:PEAK
[SOURce<HW>]:POWer:SPC:SELect
[SOURce<HW>]:POWer:SPC:STATe
[SOURce<HW>]:POWer:SPC:TARGet
[SOURce<HW>]:POWer:SPC:WARNing
```

**class SpcCls**

Spc commands group definition. 10 total commands, 2 Subgroups, 8 group commands

**get_crange**() → float

```
# SCPI: [SOURce<HW>]:POWer:SPC:CRANge
value: float = driver.source.power.spc.get_crange()
```

No command help available

**return**
pow_cntrl_crange: No help available

**get_delay**() → int

```
# SCPI: [SOURce<HW>]:POWer:SPC:DELay
value: int = driver.source.power.spc.get_delay()
```

No command help available

**return**
pow_cntrl_delay: No help available

**get_mode**() → SensorModeAll

```
# SCPI: [SOURce<HW>]:POWer:SPC:MODE
value: enums.SensorModeAll = driver.source.power.spc.get_mode()
```

No command help available

**return**
control_mode: No help available

**get_peak**() → bool

```
# SCPI: [SOURce<HW>]:POWer:SPC:PEAK
value: bool = driver.source.power.spc.get_peak()
```

No command help available

**return**
pow_cntrl_peak: No help available

**get_select**() → PowCntrlSelect

```
# SCPI: [SOURce<HW>]:POWer:SPC:SELect
value: enums.PowCntrlSelect = driver.source.power.spc.get_select()
```

No command help available

**return**
pow_cntrl_select: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:POWer:SPC:STATe
value: bool = driver.source.power.spc.get_state()
```

No command help available

**return**
pow_cntrl_state: No help available

**get_target**() → float

```
# SCPI: [SOURce<HW>]:POWer:SPC:TARGet
value: float = driver.source.power.spc.get_target()
```

No command help available

**return**
pow_cntrl_target: No help available

**get_warning_py**() → bool

```
# SCPI: [SOURce<HW>]:POWer:SPC:WARNing
value: bool = driver.source.power.spc.get_warning_py()
```

No command help available

**return**
warning_state: No help available

**set_crange**(*pow_cntrl_crange: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:CRANge
driver.source.power.spc.set_crange(pow_cntrl_crange = 1.0)
```

No command help available

**param pow_cntrl_crange**
No help available

**set_delay**(*pow_cntrl_delay: int*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:DELay
driver.source.power.spc.set_delay(pow_cntrl_delay = 1)
```

No command help available

**param pow_cntrl_delay**
No help available

**set_mode**(*control_mode: SensorModeAll*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:MODE
driver.source.power.spc.set_mode(control_mode = enums.SensorModeAll.AUTO)
```

No command help available

**param control_mode**
No help available

**set_peak**(*pow_cntrl_peak: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:PEAK
driver.source.power.spc.set_peak(pow_cntrl_peak = False)
```

No command help available

**param pow_cntrl_peak**
No help available

**set_select**(*pow_cntrl_select: PowCntrlSelect*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:SELect
driver.source.power.spc.set_select(pow_cntrl_select = enums.PowCntrlSelect.
↪SENS1)
```

No command help available

**param pow_cntrl_select**
No help available

**set_state**(*pow_cntrl_state: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:STATe
driver.source.power.spc.set_state(pow_cntrl_state = False)
```

No command help available

**param pow_cntrl_state**
No help available

**set_target**(*pow_cntrl_target: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:TARGet
driver.source.power.spc.set_target(pow_cntrl_target = 1.0)
```

No command help available

**param pow_cntrl_target**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.18.23.7.1 Measure

#### SCPI Command :

```
[SOURce<HW>]:POWer:SPC:MEASure
```

**class MeasureCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:MEASure
driver.source.power.spc.measure.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:POWer:SPC:MEASure
driver.source.power.spc.measure.set_with_opc()
```

No command help available

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

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

##### 6.18.23.7.2 Single

#### SCPI Command :

```
[SOURce<HW>]:POWer:SPC:SINGle
```

**class SingleCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:SINGle
driver.source.power.spc.single.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:POWer:SPC:SINGle
driver.source.power.spc.single.set_with_opc()
```

No command help available

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

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

#### 6.18.23.8 Step

**SCPI Commands :**

```
[SOURce<HW>]:POWer:STEP:MODE
[SOURce<HW>]:POWer:STEP:[INCRement]
```

**class StepCls**

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

**get_increment**() → float

```
# SCPI: [SOURce<HW>]:POWer:STEP:[INCRement]
value: float = driver.source.power.step.get_increment()
```

Specifies the step width in the appropriate path for POW:STEP:MODE USER. To adjust the level step-by-step with this increment value, use the command POW UP, or POW DOWN. Note: The command also sets 'Variation Step' in the manual control, that means the user-defined step width for setting the level with the rotary knob or the [Up/Down] arrow keys.

**return**
increment: float Range: 0 to 200, Unit: dB

**get_mode**() → FreqStepMode

```
# SCPI: [SOURce<HW>]:POWer:STEP:MODE
value: enums.FreqStepMode = driver.source.power.step.get_mode()
```

Defines the type of step width to vary the RF output power step-by-step with the commands POW UP or POW DOWN.

**return**
mode: DECimal| USER DECimal Increases or decreases the level in steps of ten. USER Increases or decreases the level in increments, determined with the command [:SOURcehw]:POWer:STEP[:INCRement].

**set_increment**(*increment: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:STEP:[INCRement]
driver.source.power.step.set_increment(increment = 1.0)
```

Specifies the step width in the appropriate path for POW:STEP:MODE USER. To adjust the level step-by-step with this increment value, use the command POW UP, or POW DOWN. Note: The command also sets 'Variation Step' in the manual control, that means the user-defined step width for setting the level with the rotary knob or the [Up/Down] arrow keys.

**param increment**
float Range: 0 to 200, Unit: dB

**set_mode**(*mode: FreqStepMode*) → None

```
# SCPI: [SOURce<HW>]:POWer:STEP:MODE
driver.source.power.step.set_mode(mode = enums.FreqStepMode.DECimal)
```

Defines the type of step width to vary the RF output power step-by-step with the commands POW UP or POW DOWN.

**param mode**
DECimal| USER DECimal Increases or decreases the level in steps of ten. USER Increases or decreases the level in increments, determined with the command [:SOURcehw]:POWer:STEP[:INCRement].

## 6.18.24 Psweep

**class PsweepCls**

Psweep 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.source.psweep.clone()
```

### Subgroups

#### 6.18.24.1 Trigger

**class TriggerCls**

Trigger 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.source.psweep.trigger.clone()
```

#### Subgroups

##### 6.18.24.1.1 Source

#### SCPI Command :

```
[SOURce<HW>]:PSWeep:TRIGger:SOURce:ADVanced
```

**class SourceCls**

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

**get_advanced**() → TrigSweepImmBusExt

```
# SCPI: [SOURce<HW>]:PSWeep:TRIGger:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.source.psweep.trigger.source.get_
↪advanced()
```

No command help available

**return**
ps_trig_source_adv: No help available

**set_advanced**(*ps_trig_source_adv: TrigSweepImmBusExt*) → None

```
# SCPI: [SOURce<HW>]:PSWeep:TRIGger:SOURce:ADVanced
driver.source.psweep.trigger.source.set_advanced(ps_trig_source_adv = enums.
↪TrigSweepImmBusExt.BUS)
```

No command help available

**param ps_trig_source_adv**
No help available

### 6.18.25 Pulm

#### SCPI Commands :

```
[SOURce<HW>]:PULM:DELay
[SOURce<HW>]:PULM:IMPedance
[SOURce<HW>]:PULM:MODE
[SOURce<HW>]:PULM:PERiod
[SOURce<HW>]:PULM:POLarity
[SOURce<HW>]:PULM:SOURce
[SOURce<HW>]:PULM:STATe
[SOURce<HW>]:PULM:THReshold
[SOURce<HW>]:PULM:TTYPe
[SOURce<HW>]:PULM:WIDTh
```

**class PulmCls**

Pulm commands group definition. 46 total commands, 4 Subgroups, 10 group commands

**get_delay**() → float

```
# SCPI: [SOURce<HW>]:PULM:DELay
value: float = driver.source.pulm.get_delay()
```

Sets the pulse delay.

**return**
delay: float

**get_impedance**() → InpImpRf

```
# SCPI: [SOURce<HW>]:PULM:IMPedance
value: enums.InpImpRf = driver.source.pulm.get_impedance()
```

Sets the impedance for the external pulse trigger and pulse modulation input.

**return**
impedance: G50| G10K

**get_mode**() → PulsMode

```
# SCPI: [SOURce<HW>]:PULM:MODE
value: enums.PulsMode = driver.source.pulm.get_mode()
```

Selects the mode for the pulse modulation.

**return**
mode: SINGle| DOUBle | PTRain SINGle Generates a single pulse. DOUBle Generates two pulses within one pulse period. PTRain Generates a user-defined pulse train. Specify the pulse sequence with the commands: [:SOURcehw]:PULM:TRAin:ONTime [:SOURcehw]:PULM:TRAin:OFFTime [:SOURcehw]:PULM:TRAin:REPetition

**get_period**() → float

```
# SCPI: [SOURce<HW>]:PULM:PERiod
value: float = driver.source.pulm.get_period()
```

Sets the period of the generated pulse, that means the repetition frequency of the internally generated modulation signal.

**return**
period: float The minimum value depends on the installed options R&S SMAB-K22 or R&S SMAB-K23 Range: 20E-9 to 100

**get_polarity**() → NormalInverted

```
# SCPI: [SOURce<HW>]:PULM:POLarity
value: enums.NormalInverted = driver.source.pulm.get_polarity()
```

Sets the polarity of the externally applied modulation signal.

**return**
polarity: NORMal| INVerted NORMal Suppresses the RF signal during the pulse pause. INVerted Suppresses the RF signal during the pulse.

**get_source**() → SourceInt

```
# SCPI: [SOURce<HW>]:PULM:SOURce
value: enums.SourceInt = driver.source.pulm.get_source()
```

Selects between the internal (pulse generator) or an external pulse signal for the modulation.

**return**
source: INTernal| EXTernal

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:PULM:STATe
value: bool = driver.source.pulm.get_state()
```

Activates pulse modulation.

**return**
state: 1| ON| 0| OFF

**get_threshold**() → float

```
# SCPI: [SOURce<HW>]:PULM:THReshold
value: float = driver.source.pulm.get_threshold()
```

Sets the threshold for the input signal at the [Pulse Ext] connector.

**return**
threshold: float Range: 0 to 2, Unit: V

**get_ttype**() → PulsTransType

```
# SCPI: [SOURce<HW>]:PULM:TTYPe
value: enums.PulsTransType = driver.source.pulm.get_ttype()
```

Sets the transition mode for the pulse signal.

**return**
source: SMOothed| FAST SMOothed flattens the slew rate, resulting in longer rise/fall times. FAST enables fast transitions with shortest rise and fall times.

**get_width**() → float

```
# SCPI: [SOURce<HW>]:PULM:WIDTh
value: float = driver.source.pulm.get_width()
```

Sets the width of the generated pulse, that means the pulse length. It must be at least 20ns less than the set pulse period.

**return**
width: float Range: 20E-9 to 100

**set_delay**(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:DELay
driver.source.pulm.set_delay(delay = 1.0)
```

Sets the pulse delay.

**param delay**
float

**set_impedance**(*impedance: InpImpRf*) → None

```
# SCPI: [SOURce<HW>]:PULM:IMPedance
driver.source.pulm.set_impedance(impedance = enums.InpImpRf.G10K)
```

Sets the impedance for the external pulse trigger and pulse modulation input.

**param impedance**
G50| G10K

**set_mode**(*mode: PulsMode*) → None

```
# SCPI: [SOURce<HW>]:PULM:MODE
driver.source.pulm.set_mode(mode = enums.PulsMode.DOUBle)
```

Selects the mode for the pulse modulation.

**param mode**
SINGle| DOUBle | PTRain SINGle Generates a single pulse. DOUBle Generates two pulses within one pulse period. PTRain Generates a user-defined pulse train. Specify the pulse sequence with the commands: [:SOURcehw]:PULM:TRAin:ONTime [:SOURcehw]:PULM:TRAin:OFFTime [:SOURcehw]:PULM:TRAin:REPetition

**set_period**(*period: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:PERiod
driver.source.pulm.set_period(period = 1.0)
```

Sets the period of the generated pulse, that means the repetition frequency of the internally generated modulation signal.

**param period**
float The minimum value depends on the installed options R&S SMAB-K22 or R&S SMAB-K23 Range: 20E-9 to 100

**set_polarity**(*polarity: NormalInverted*) → None

```
# SCPI: [SOURce<HW>]:PULM:POLarity
driver.source.pulm.set_polarity(polarity = enums.NormalInverted.INVerted)
```

Sets the polarity of the externally applied modulation signal.

**param polarity**
NORMal| INVerted NORMal Suppresses the RF signal during the pulse pause. INVerted Suppresses the RF signal during the pulse.

**set_source**(*source: SourceInt*) → None

```
# SCPI: [SOURce<HW>]:PULM:SOURce
driver.source.pulm.set_source(source = enums.SourceInt.EXTernal)
```

Selects between the internal (pulse generator) or an external pulse signal for the modulation.

**param source**
INTernal| EXTernal

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:PULM:STATe
driver.source.pulm.set_state(state = False)
```

Activates pulse modulation.

**param state**
1| ON| 0| OFF

**set_threshold**(*threshold: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:THReshold
driver.source.pulm.set_threshold(threshold = 1.0)
```

Sets the threshold for the input signal at the [Pulse Ext] connector.

**param threshold**
float Range: 0 to 2, Unit: V

**set_ttype**(*source: PulsTransType*) → None

```
# SCPI: [SOURce<HW>]:PULM:TTYPe
driver.source.pulm.set_ttype(source = enums.PulsTransType.FAST)
```

Sets the transition mode for the pulse signal.

**param source**
SMOothed| FAST SMOothed flattens the slew rate, resulting in longer rise/fall times.
FAST enables fast transitions with shortest rise and fall times.

**set_width**(*width: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:WIDTh
driver.source.pulm.set_width(width = 1.0)
```

Sets the width of the generated pulse, that means the pulse length. It must be at least 20ns less than the set pulse period.

**param width**
float Range: 20E-9 to 100

### Cloning the Group

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

### Subgroups

#### 6.18.25.1 Double

### SCPI Commands :

```
[SOURce<HW>]:PULM:DOUBle:DELay
[SOURce<HW>]:PULM:DOUBle:STATe
[SOURce<HW>]:PULM:DOUBle:WIDTh
```

**class DoubleCls**

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

**get_delay**() → float

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:DELay
value: float = driver.source.pulm.double.get_delay()
```

Sets the delay from the start of the first pulse to the start of the second pulse.

**return**
delay: float

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:STATe
value: bool = driver.source.pulm.double.get_state()
```

Provided for backward compatibility with former Rohde & Schwarz signal generators. Works like the command [:SOURce<hw>]:PULM:MODEDOUBle.

**return**
state: 1| ON| 0| OFF

**get_width**() → float

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:WIDTh
value: float = driver.source.pulm.double.get_width()
```

Sets the width of the second pulse.

**return**
width: float

**set_delay**(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:DELay
driver.source.pulm.double.set_delay(delay = 1.0)
```

Sets the delay from the start of the first pulse to the start of the second pulse.

**param delay**
float

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:STATe
driver.source.pulm.double.set_state(state = False)
```

Provided for backward compatibility with former Rohde & Schwarz signal generators. Works like the command [:SOURce<hw>]:PULM:MODEDOUBle.

**param state**
1| ON| 0| OFF

**set_width**(*width: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:WIDTh
driver.source.pulm.double.set_width(width = 1.0)
```

Sets the width of the second pulse.

**param width**
float

#### 6.18.25.2 Internal

**class InternalCls**

Internal 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.source.pulm.internal.clone()
```

##### Subgroups

###### 6.18.25.2.1 Train

**class TrainCls**

Train 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.source.pulm.internal.train.clone()
```

### Subgroups

#### 6.18.25.2.1.1 Trigger

**class TriggerCls**

Trigger 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.source.pulm.internal.train.trigger.clone()
```

### Subgroups

#### 6.18.25.2.1.2 Immediate

### SCPI Command :

```
[SOURce]:PULM:[INTernal]:[TRAin]:TRIGger:IMMediate
```

**class ImmediateCls**

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

**set**() → None

```
# SCPI: [SOURce]:PULM:[INTernal]:[TRAin]:TRIGger:IMMediate
driver.source.pulm.internal.train.trigger.immediate.set()
```

If [:SOURce<hw>]:PULM:TRIGger:MODESINGle, triggers the pulse generator.

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

```
# SCPI: [SOURce]:PULM:[INTernal]:[TRAin]:TRIGger:IMMediate
driver.source.pulm.internal.train.trigger.immediate.set_with_opc()
```

If [:SOURce<hw>]:PULM:TRIGger:MODESINGle, triggers the pulse generator.

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

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

#### 6.18.25.3 Train

**SCPI Commands :**

```
[SOURce<HW>]:PULM:TRAin:CATalog
[SOURce<HW>]:PULM:TRAin:DELete
[SOURce<HW>]:PULM:TRAin:SELect
```

**class TrainCls**

Train commands group definition. 30 total commands, 5 Subgroups, 3 group commands

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DELete
driver.source.pulm.train.delete(filename = 'abc')
```

Deletes the specified pulse train file. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
string Filename or complete file path; file extension is optional.

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:CATalog
value: List[str] = driver.source.pulm.train.get_catalog()
```

Queries the available pulse train files in the specified directory.

**return**
catalog: string List of list filenames, separated by commas

**get_select**() → str

```
# SCPI: [SOURce<HW>]:PULM:TRAin:SELect
value: str = driver.source.pulm.train.get_select()
```

Selects or creates a data list in pulse train mode. If the list with the selected name does not exist, a new list is created.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:SELect
driver.source.pulm.train.set_select(filename = 'abc')
```

Selects or creates a data list in pulse train mode. If the list with the selected name does not exist, a new list is created.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

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

### Subgroups

#### 6.18.25.3.1 Dexchange

### SCPI Commands :

```
[SOURce<HW>]:PULM:TRAin:DEXChange:MODE
[SOURce<HW>]:PULM:TRAin:DEXChange:SELect
```

**class DexchangeCls**

Dexchange commands group definition. 8 total commands, 2 Subgroups, 2 group commands

**get_mode**() → DexchMode

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:MODE
value: enums.DexchMode = driver.source.pulm.train.dexchange.get_mode()
```

Determines the import or export of a list. Specify the source or destination file with the command [:SOURce<hw>]:PULM:TRAin:DEXChange:SELect.

**return**
mode: IMPort| EXPort

**get_select**() → str

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:SELect
value: str = driver.source.pulm.train.dexchange.get_select()
```

Selects the ASCII file for import or export, containing a pulse train list.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_mode**(*mode: DexchMode*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:MODE
driver.source.pulm.train.dexchange.set_mode(mode = enums.DexchMode.EXPort)
```

Determines the import or export of a list. Specify the source or destination file with the command [:SOURce<hw>]:PULM:TRAin:DEXChange:SELect.

**param mode**
IMPort| EXPort

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:SELect
driver.source.pulm.train.dexchange.set_select(filename = 'abc')
```

Selects the ASCII file for import or export, containing a pulse train list.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.pulm.train.dexchange.clone()
```

### Subgroups

#### 6.18.25.3.1.1 Afile

### SCPI Commands :

```
[SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:CATalog
[SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:EXTension
[SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SELect
```

**class AfileCls**

Afile commands group definition. 5 total commands, 1 Subgroups, 3 group commands

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:CATalog
value: List[str] = driver.source.pulm.train.dexchange.afile.get_catalog()
```

Queries the available ASCII files in the current or specified directory.

**return**
catalog: string List of ASCII files *.txt or *.csv, separated by commas.

**get_extension**() → DexchExtension

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:EXTension
value: enums.DexchExtension = driver.source.pulm.train.dexchange.afile.get_
↪extension()
```

Determines the extension of the ASCII file for import or export, or to query existing files.

**return**
extension: TXT| CSV

**get_select**() → str

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SELect
value: str = driver.source.pulm.train.dexchange.afile.get_select()
```

Selects the ASCII file to be imported or exported.

**return**
filename: string Filename or complete file path; file extension can be omitted.

**set_extension**(*extension: DexchExtension*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:EXTension
driver.source.pulm.train.dexchange.afile.set_extension(extension = enums.
↪DexchExtension.CSV)
```

Determines the extension of the ASCII file for import or export, or to query existing files.

**param extension**
TXT| CSV

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SELect
driver.source.pulm.train.dexchange.afile.set_select(filename = 'abc')
```

Selects the ASCII file to be imported or exported.

**param filename**
string Filename or complete file path; file extension can be omitted.

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.pulm.train.dexchange.afile.clone()
```

### Subgroups

#### 6.18.25.3.1.2 Separator

### SCPI Commands :

```
[SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SEParator:COLumn
[SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SEParator:DECimal
```

**class SeparatorCls**

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

**get_column**() → DexchSepCol

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SEParator:COLumn
value: enums.DexchSepCol = driver.source.pulm.train.dexchange.afile.separator.
↪get_column()
```

Selects the separator between the frequency and level column of the ASCII table.

**return**
column: TABulator| SEMicolon| COMMa| SPACe

**get_decimal**() → DecimalSeparator

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SEParator:DECimal
value: enums.DecimalSeparator = driver.source.pulm.train.dexchange.afile.
↪separator.get_decimal()
```

Sets '.' (decimal point) or ',' (comma) as the decimal separator used in the ASCII data with floating-point numerals.

**return**
decimal: DOT| COMMa

**set_column**(*column: DexchSepCol*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SEParator:COLumn
driver.source.pulm.train.dexchange.afile.separator.set_column(column = enums.
↪DexchSepCol.COMMa)
```

Selects the separator between the frequency and level column of the ASCII table.

**param column**
TABulator| SEMicolon| COMMa| SPACe

**set_decimal**(*decimal: DecimalSeparator*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:AFILe:SEParator:DECimal
driver.source.pulm.train.dexchange.afile.separator.set_decimal(decimal = enums.
↪DecimalSeparator.COMMa)
```

Sets '.' (decimal point) or ',' (comma) as the decimal separator used in the ASCII data with floating-point numerals.

**param decimal**
DOT| COMMa

#### 6.18.25.3.1.3 Execute

**SCPI Command :**

```
[SOURce<HW>]:PULM:TRAin:DEXChange:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:EXECute
driver.source.pulm.train.dexchange.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:PULM:TRAin:DEXChange:EXECute
driver.source.pulm.train.dexchange.execute.set_with_opc()
```

No command help available

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

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

#### 6.18.25.3.2 Hopping

**SCPI Commands :**

```
[SOURce<HW>]:PULM:TRAin:HOPPing:CATalog
[SOURce<HW>]:PULM:TRAin:HOPPing:DELete
[SOURce<HW>]:PULM:TRAin:HOPPing:SELect
```

**class HoppingCls**

Hopping commands group definition. 13 total commands, 5 Subgroups, 3 group commands

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:DELete
driver.source.pulm.train.hopping.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:CATalog
value: List[str] = driver.source.pulm.train.hopping.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_select**() → str

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:SELect
value: str = driver.source.pulm.train.hopping.get_select()
```

No command help available

**return**
filename: No help available

**set_select**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:SELect
driver.source.pulm.train.hopping.set_select(filename = 'abc')
```

No command help available

**param filename**
No help available

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.pulm.train.hopping.clone()
```

### Subgroups

#### 6.18.25.3.2.1 Frequency

### SCPI Commands :

```
[SOURce<HW>]:PULM:TRAin:HOPPing:FREQuency:POINts
[SOURce<HW>]:PULM:TRAin:HOPPing:FREQuency
```

**class FrequencyCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:FREQuency:POINts
value: int = driver.source.pulm.train.hopping.frequency.get_points()
```

No command help available

**return**
points: No help available

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:FREQuency
value: List[float] = driver.source.pulm.train.hopping.frequency.get_value()
```

No command help available

**return**
frequency: No help available

**set_value**(*frequency: List[float]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:FREQuency
driver.source.pulm.train.hopping.frequency.set_value(frequency = [1.1, 2.2, 3.
↪3])
```

No command help available

**param frequency**
No help available

#### 6.18.25.3.2.2 OffTime

**SCPI Commands :**

```
[SOURce<HW>]:PULM:TRAin:HOPPing:OFFTime:POINts
[SOURce<HW>]:PULM:TRAin:HOPPing:OFFTime
```

**class OffTimeCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:OFFTime:POINts
value: int = driver.source.pulm.train.hopping.offTime.get_points()
```

No command help available

**return**
points: No help available

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:OFFTime
value: List[float] = driver.source.pulm.train.hopping.offTime.get_value()
```

No command help available

**return**
off_time: No help available

**set_value**(*off_time: List[float]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:OFFTime
driver.source.pulm.train.hopping.offTime.set_value(off_time = [1.1, 2.2, 3.3])
```

No command help available

**param off_time**
No help available

#### 6.18.25.3.2.3 Ontime

**SCPI Commands :**

```
[SOURce<HW>]:PULM:TRAin:HOPPing:ONTime:POINts
[SOURce<HW>]:PULM:TRAin:HOPPing:ONTime
```

**class OntimeCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:ONTime:POINts
value: int = driver.source.pulm.train.hopping.ontime.get_points()
```

No command help available

**return**
points: No help available

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:ONTime
value: List[float] = driver.source.pulm.train.hopping.ontime.get_value()
```

No command help available

**return**
ontime: No help available

**set_value**(*ontime: List[float]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:ONTime
driver.source.pulm.train.hopping.ontime.set_value(ontime = [1.1, 2.2, 3.3])
```

No command help available

**param ontime**
No help available

#### 6.18.25.3.2.4 Power

##### SCPI Commands :

```
[SOURce<HW>]:PULM:TRAin:HOPPing:POWer:POINts
[SOURce<HW>]:PULM:TRAin:HOPPing:POWer
```

**class PowerCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:POWer:POINts
value: int = driver.source.pulm.train.hopping.power.get_points()
```

No command help available

**return**
points: No help available

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:POWer
value: List[float] = driver.source.pulm.train.hopping.power.get_value()
```

No command help available

**return**
power: No help available

**set_value**(*power: List[float]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:POWer
driver.source.pulm.train.hopping.power.set_value(power = [1.1, 2.2, 3.3])
```

No command help available

**param power**
No help available

#### 6.18.25.3.2.5 Repetition

##### SCPI Commands :

```
[SOURce<HW>]:PULM:TRAin:HOPPing:REPetition:POINts
[SOURce<HW>]:PULM:TRAin:HOPPing:REPetition
```

**class RepetitionCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:REPetition:POINts
value: int = driver.source.pulm.train.hopping.repetition.get_points()
```

No command help available

**return**
points: No help available

**get_value**() → List[int]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:REPetition
value: List[int] = driver.source.pulm.train.hopping.repetition.get_value()
```

No command help available

**return**
repetition: No help available

**set_value**(*repetition: List[int]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:HOPPing:REPetition
driver.source.pulm.train.hopping.repetition.set_value(repetition = [1, 2, 3])
```

No command help available

**param repetition**
No help available

#### 6.18.25.3.3 OffTime

**SCPI Commands :**

```
[SOURce<HW>]:PULM:TRAin:OFFTime:POINts
[SOURce<HW>]:PULM:TRAin:OFFTime
```

**class OffTimeCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:OFFTime:POINts
value: int = driver.source.pulm.train.offTime.get_points()
```

Queries the number of on and off time entries and repetitions in the selected list.

**return**
points: integer Range: 0 to INT_MAX

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:OFFTime
value: List[float] = driver.source.pulm.train.offTime.get_value()
```

Enters the pulse on/off times values in the selected list.

**return**
off_time: Offtime#1{, Offtime#2, ...} | binary block data List of comma-separated numeric values or binary block data, where: The list of numbers can be of any length. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See method RsSmab.FormatPy.data for details. The maximum length is 2047 values. Range: 0 ns to 5 ms

**set_value**(*off_time: List[float]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:OFFTime
driver.source.pulm.train.offTime.set_value(off_time = [1.1, 2.2, 3.3])
```

Enters the pulse on/off times values in the selected list.

**param off_time**
Offtime#1{, Offtime#2, ...} | binary block data List of comma-separated numeric values or binary block data, where: The list of numbers can be of any length. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See method RsSmab.FormatPy.data for details. The maximum length is 2047 values. Range: 0 ns to 5 ms

#### 6.18.25.3.4 Ontime

**SCPI Commands :**

```
[SOURce<HW>]:PULM:TRAin:ONTime:POINts
[SOURce<HW>]:PULM:TRAin:ONTime
```

**class OntimeCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:ONTime:POINts
value: int = driver.source.pulm.train.ontime.get_points()
```

Queries the number of on and off time entries and repetitions in the selected list.

**return**
points: integer Range: 0 to INT_MAX

**get_value**() → List[float]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:ONTime
value: List[float] = driver.source.pulm.train.ontime.get_value()
```

Enters the pulse on/off times values in the selected list.

**return**
ontime: No help available

**set_value**(*ontime: List[float]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:ONTime
driver.source.pulm.train.ontime.set_value(ontime = [1.1, 2.2, 3.3])
```

Enters the pulse on/off times values in the selected list.

**param ontime**
Offtime#1{, Offtime#2, ... } | binary block data List of comma-separated numeric values or binary block data, where: The list of numbers can be of any length. In binary block format, 8 (4) bytes are always interpreted as a floating-point number with double accuracy. See method RsSmab.FormatPy.data for details. The maximum length is 2047 values. Range: 0 ns to 5 ms

#### 6.18.25.3.5 Repetition

**SCPI Commands :**

```
[SOURce<HW>]:PULM:TRAin:REPetition:POINts
[SOURce<HW>]:PULM:TRAin:REPetition
```

**class RepetitionCls**

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

**get_points**() → int

```
# SCPI: [SOURce<HW>]:PULM:TRAin:REPetition:POINts
value: int = driver.source.pulm.train.repetition.get_points()
```

Queries the number of on and off time entries and repetitions in the selected list.

**return**
points: integer Range: 0 to INT_MAX

**get_value**() → List[int]

```
# SCPI: [SOURce<HW>]:PULM:TRAin:REPetition
value: List[int] = driver.source.pulm.train.repetition.get_value()
```

Sets the number of repetitions for each pulse on/off time value pair.

**return**
repetition: Repetition#1{, Repetition#2, ... } 0 = ignore value pair Set 'Repetition = 0' to skip a particular pulse without deleting the pulse on/off time value pair Range: 0 to 65535

**set_value**(*repetition: List[int]*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRAin:REPetition
driver.source.pulm.train.repetition.set_value(repetition = [1, 2, 3])
```

Sets the number of repetitions for each pulse on/off time value pair.

**param repetition**
Repetition#1{, Repetition#2, ... } 0 = ignore value pair Set 'Repetition = 0' to skip a particular pulse without deleting the pulse on/off time value pair Range: 0 to 65535

#### 6.18.25.4 Trigger

**SCPI Command :**

```
[SOURce<HW>]:PULM:TRIGger:MODE
```

**class TriggerCls**

Trigger commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_mode**() → PulsTrigModeWithSingle

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:MODE
value: enums.PulsTrigModeWithSingle = driver.source.pulm.trigger.get_mode()
```

Selects a trigger mode - auto, single, external, external single or external gated - for generating the modulation signal.

**return**
mode: AUTO| EXTernal| EGATe| SINGle| ESINgle

**set_mode**(*mode: PulsTrigModeWithSingle*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:MODE
driver.source.pulm.trigger.set_mode(mode = enums.PulsTrigModeWithSingle.AUTO)
```

Selects a trigger mode - auto, single, external, external single or external gated - for generating the modulation signal.

**param mode**
AUTO| EXTernal| EGATe| SINGle| ESINgle

### Cloning the Group

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

### Subgroups

#### 6.18.25.4.1 External

### SCPI Command :

```
[SOURce<HW>]:PULM:TRIGger:EXTernal:IMPedance
```

**class ExternalCls**

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

**get_impedance**() → InpImpRf

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:IMPedance
value: enums.InpImpRf = driver.source.pulm.trigger.external.get_impedance()
```

No command help available

**return**
impedance: No help available

**set_impedance**(*impedance: InpImpRf*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:IMPedance
driver.source.pulm.trigger.external.set_impedance(impedance = enums.InpImpRf.
↪G10K)
```

No command help available

**param impedance**
No help available

### 6.18.26 Roscillator

#### SCPI Commands :

```
[SOURce]:ROSCillator:PRESet
[SOURce]:ROSCillator:SOURce
```

**class RoscillatorCls**

Roscillator commands group definition. 14 total commands, 3 Subgroups, 2 group commands

**get_source**() → SourceInt

```
# SCPI: [SOURce]:ROSCillator:SOURce
value: enums.SourceInt = driver.source.roscillator.get_source()
```

Selects between internal or external reference frequency.

**return**
source: INTernal| EXTernal

**preset**() → None

```
# SCPI: [SOURce]:ROSCillator:PRESet
driver.source.roscillator.preset()
```

Resets the reference oscillator settings.

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

```
# SCPI: [SOURce]:ROSCillator:PRESet
driver.source.roscillator.preset_with_opc()
```

Resets the reference oscillator settings.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_source**(*source: SourceInt*) → None

```
# SCPI: [SOURce]:ROSCillator:SOURce
driver.source.roscillator.set_source(source = enums.SourceInt.EXTernal)
```

Selects between internal or external reference frequency.

**param source**
INTernal| EXTernal

### Cloning the Group

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

### Subgroups

#### 6.18.26.1 External

### SCPI Commands :

```
[SOURce]:ROSCillator:EXTernal:MLRange
[SOURce]:ROSCillator:EXTernal:NSBandwidth
[SOURce]:ROSCillator:EXTernal:SBANdwidth
```

**class ExternalCls**

External commands group definition. 6 total commands, 2 Subgroups, 3 group commands

**get_mlrange**() → str

```
# SCPI: [SOURce]:ROSCillator:EXTernal:MLRange
value: str = driver.source.roscillator.external.get_mlrange()
```

Queries the minimum locking range for the selected external reference frequency. Depending on the RF hardware version, and the installed options, the minimum locking range vaies. For more information, see data sheet.

**return**
min_lock_range: string

**get_ns_bandwidth**() → str

```
# SCPI: [SOURce]:ROSCillator:EXTernal:NSBandwidth
value: str = driver.source.roscillator.external.get_ns_bandwidth()
```

Queries the nominal synchronization bandwidth for the selected external reference frequency and synchronization bandwidth.

**return**
nom_bandwidth: string

**get_sbandwidth**() → FilterWidth

```
# SCPI: [SOURce]:ROSCillator:EXTernal:SBANdwidth
value: enums.FilterWidth = driver.source.roscillator.external.get_sbandwidth()
```

Selects the synchronization bandwidth for the external reference signal. See [:SOURce]:ROSCillator:SOURce > External. Depending on the RF hardware version, and the installed options, the synchronization bandwidth varies. For more information, see data sheet.

**return**
sbandwidth: WIDE| NARRow NARRow The synchronization bandwidth is a few Hz.
WIDE Uses the widest possible synchronization bandwidth.

**set_sbandwidth**(*sbandwidth: FilterWidth*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:SBANdwidth
driver.source.roscillator.external.set_sbandwidth(sbandwidth = enums.
↪FilterWidth.NARRow)
```

Selects the synchronization bandwidth for the external reference signal. See [:SOURce]:ROSCillator:SOURce > External. Depending on the RF hardware version, and the installed options, the synchronization bandwidth varies. For more information, see data sheet.

**param sbandwidth**
WIDE| NARRow NARRow The synchronization bandwidth is a few Hz. WIDE Uses the widest possible synchronization bandwidth.

### Cloning the Group

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

### Subgroups

#### 6.18.26.1.1 Frequency

### SCPI Commands :

```
[SOURce]:ROSCillator:EXTernal:FREQuency:VARiable
[SOURce]:ROSCillator:EXTernal:FREQuency
```

**class FrequencyCls**

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

**get_value**() → RoscFreqExt

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency
value: enums.RoscFreqExt = driver.source.roscillator.external.frequency.get_
↪value()
```

Sets the frequency of the external reference.

**return**
frequency: 100MHZ| 1GHZ| VARiable| 10MHZ

**get_variable**() → float

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency:VARiable
value: float = driver.source.roscillator.external.frequency.get_variable()
```

Specifies the user-defined external reference frequency.

**return**
frequency: float Range: 1E6 to 100E6, Unit: Hz

**set_value**(*frequency: RoscFreqExt*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency
driver.source.roscillator.external.frequency.set_value(frequency = enums.
↪RoscFreqExt._100MHZ)
```

Sets the frequency of the external reference.

**param frequency**
100MHZ| 1GHZ| VARiable| 10MHZ

**set_variable**(*frequency: float*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency:VARiable
driver.source.roscillator.external.frequency.set_variable(frequency = 1.0)
```

Specifies the user-defined external reference frequency.

**param frequency**
float Range: 1E6 to 100E6, Unit: Hz

#### 6.18.26.1.2 RfOff

**SCPI Command :**

```
[SOURce]:ROSCillator:EXTernal:RFOFf:[STATe]
```

**class RfOffCls**

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

**get_state**() → bool

```
# SCPI: [SOURce]:ROSCillator:EXTernal:RFOFf:[STATe]
value: bool = driver.source.roscillator.external.rfOff.get_state()
```

Determines that the RF output is turned off when the external reference signal is selected, but missing.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:RFOFf:[STATe]
driver.source.roscillator.external.rfOff.set_state(state = False)
```

Determines that the RF output is turned off when the external reference signal is selected, but missing.

**param state**
1| ON| 0| OFF

#### 6.18.26.2 Internal

**class InternalCls**

Internal 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.source.roscillator.internal.clone()
```

### Subgroups

#### 6.18.26.2.1 Adjust

### SCPI Commands :

```
[SOURce]:ROSCillator:[INTernal]:ADJust:VALue
[SOURce]:ROSCillator:[INTernal]:ADJust:[STATe]
```

**class AdjustCls**

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

**get_state**() → bool

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:[STATe]
value: bool = driver.source.roscillator.internal.adjust.get_state()
```

Determines whether the calibrated (off) or a user-defined (on) adjustment value is used for fine adjustment of the frequency.

**return**
state: 1| ON| 0| OFF 0 Fine adjustment with the calibrated frequency value 1 User-defined adjustment value. The instrument is no longer in the calibrated state. The calibration value is, however, not changed. The instrument resumes the calibrated state if you send SOURce:ROSCillator:INTernal:ADJust:STATe 0.

**get_value**() → int

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:VALue
value: int = driver.source.roscillator.internal.adjust.get_value()
```

Specifies the frequency correction value (adjustment value) .

**return**
value: integer

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:[STATe]
driver.source.roscillator.internal.adjust.set_state(state = False)
```

Determines whether the calibrated (off) or a user-defined (on) adjustment value is used for fine adjustment of the frequency.

**param state**
1| ON| 0| OFF 0 Fine adjustment with the calibrated frequency value 1 User-defined adjustment value. The instrument is no longer in the calibrated state. The calibration value is, however, not changed. The instrument resumes the calibrated state if you send SOURce:ROSCillator:INTernal:ADJust:STATe 0.

**set_value**(*value: int*) → None

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:VALue
driver.source.roscillator.internal.adjust.set_value(value = 1)
```

Specifies the frequency correction value (adjustment value) .

**param value**
integer

#### 6.18.26.2.2 Tuning

**SCPI Commands :**

```
[SOURce]:ROSCillator:INTernal:TUNing:SLOPe
[SOURce]:ROSCillator:INTernal:TUNing:[STATe]
```

**class TuningCls**

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

**get_slope**() → LowHigh

```
# SCPI: [SOURce]:ROSCillator:INTernal:TUNing:SLOPe
value: enums.LowHigh = driver.source.roscillator.internal.tuning.get_slope()
```

Sets the sensitivity of the external tuning volatge.

**return**
state: LOW| HIGH

**get_state**() → bool

```
# SCPI: [SOURce]:ROSCillator:INTernal:TUNing:[STATe]
value: bool = driver.source.roscillator.internal.tuning.get_state()
```

Activates the EFC (external frequency control) .

**return**
state: 1| ON| 0| OFF

**set_slope**(*state: LowHigh*) → None

```
# SCPI: [SOURce]:ROSCillator:INTernal:TUNing:SLOPe
driver.source.roscillator.internal.tuning.set_slope(state = enums.LowHigh.HIGH)
```

Sets the sensitivity of the external tuning volatge.

**param state**
LOW| HIGH

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce]:ROSCillator:INTernal:TUNing:[STATe]
driver.source.roscillator.internal.tuning.set_state(state = False)
```

Activates the EFC (external frequency control) .

**param state**
1| ON| 0| OFF

#### 6.18.26.3 Output

**class OutputCls**

Output 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.source.roscillator.output.clone()
```

##### Subgroups

#### 6.18.26.3.1 Alternate

**class AlternateCls**

Alternate 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.source.roscillator.output.alternate.clone()
```

##### Subgroups

#### 6.18.26.3.1.1 Frequency

##### SCPI Command :

```
[SOURce]:ROSCillator:OUTPut:ALTernate:FREQuency:MODE
```

**class FrequencyCls**

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

**get_mode**() → Rosc1GoUtpFreqMode

```
# SCPI: [SOURce]:ROSCillator:OUTPut:ALTernate:FREQuency:MODE
value: enums.Rosc1GoUtpFreqMode = driver.source.roscillator.output.alternate.
↪frequency.get_mode()
```

Sets the output reference frequency.

**return**
outp_freq_mode: LOOPthrough| DER1G| OFF OFF Disables the output. DER1G Sets the output reference frequency to 1 GHz. The reference frequency is derived from the internal reference frequency. LOOPthrough If [:SOURce]:ROSCillator:EXTernal:FREQuency 1GHZ, forwards the input reference frequency to the reference frequency output.

**set_mode**(*outp_freq_mode: Rosc1GoUtpFreqMode*) → None

```
# SCPI: [SOURce]:ROSCillator:OUTPut:ALTernate:FREQuency:MODE
driver.source.roscillator.output.alternate.frequency.set_mode(outp_freq_mode =␣
↪enums.Rosc1GoUtpFreqMode.DER1G)
```

Sets the output reference frequency.

**param outp_freq_mode**
LOOPthrough| DER1G| OFF OFF Disables the output. DER1G Sets the output reference frequency to 1 GHz. The reference frequency is derived from the internal reference frequency. LOOPthrough If [:SOURce]:ROSCillator:EXTernal:FREQuency 1GHZ, forwards the input reference frequency to the reference frequency output.

#### 6.18.26.3.2 Frequency

**SCPI Command :**

```
[SOURce]:ROSCillator:OUTPut:FREQuency:MODE
```

**class FrequencyCls**

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

**get_mode**() → RoscOutpFreqMode

```
# SCPI: [SOURce]:ROSCillator:OUTPut:FREQuency:MODE
value: enums.RoscOutpFreqMode = driver.source.roscillator.output.frequency.get_
↪mode()
```

Sets the output reference frequency.

**return**
outp_freq_mode: DER10M| DER100M| OFF| LOOPthrough OFF Disables the output. DER10M|DER100M Sets the output reference frequency to 10 MHz or 100 MHz. The reference frequency is derived from the internal reference frequency. LOOPthrough This option is unavailable for ROSCillator:EXTernal:FREQuency 1GHZ. Forwards the input reference frequency to the reference frequency output.

**set_mode**(*outp_freq_mode: RoscOutpFreqMode*) → None

```
# SCPI: [SOURce]:ROSCillator:OUTPut:FREQuency:MODE
driver.source.roscillator.output.frequency.set_mode(outp_freq_mode = enums.
↪RoscOutpFreqMode.DER100M)
```

Sets the output reference frequency.

**param outp_freq_mode**
DER10M| DER100M| OFF| LOOPthrough OFF Disables the output. DER10M|DER100M Sets the output reference frequency to 10 MHz or 100 MHz. The reference frequency is derived from the internal reference frequency. LOOPthrough This option is unavailable for ROSCillator:EXTernal:FREQuency 1GHZ. Forwards the input reference frequency to the reference frequency output.

### 6.18.27 Sweep

#### SCPI Commands :

```
[SOURce<HW>]:SWEep:GENeration
[SOURce<HW>]:SWEep:RESet:[ALL]
```

**class SweepCls**

Sweep commands group definition. 36 total commands, 4 Subgroups, 2 group commands

**get_generation**() → FreqSweepType

```
# SCPI: [SOURce<HW>]:SWEep:GENeration
value: enums.FreqSweepType = driver.source.sweep.get_generation()
```

Selects frequency sweep type.

**return**
sweep_type: STEPped| ANALog STEPped Performs a frequency sweep. ANALog Performs a continuous analog frequency sweep (ramp) , synchronized with the sweep time [:SOURcehw]:SWEep[:FREQuency]:TIME.

**reset_all**() → None

```
# SCPI: [SOURce<HW>]:SWEep:RESet:[ALL]
driver.source.sweep.reset_all()
```

Resets all active sweeps to the starting point.

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

```
# SCPI: [SOURce<HW>]:SWEep:RESet:[ALL]
driver.source.sweep.reset_all_with_opc()
```

Resets all active sweeps to the starting point.

Same as reset_all, but waits for the operation to complete before continuing further. Use the RsSmab.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_generation**(*sweep_type: FreqSweepType*) → None

```
# SCPI: [SOURce<HW>]:SWEep:GENeration
driver.source.sweep.set_generation(sweep_type = enums.FreqSweepType.ANALog)
```

Selects frequency sweep type.

**param sweep_type**
STEPped| ANALog STEPped Performs a frequency sweep. ANALog Performs a continuous analog frequency sweep (ramp) , synchronized with the sweep time [:SOURcehw]:SWEep[:FREQuency]:TIME.

### Cloning the Group

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

### Subgroups

#### 6.18.27.1 Combined

### SCPI Commands :

```
[SOURce<HW>]:SWEep:COMBined:COUNt
[SOURce<HW>]:SWEep:COMBined:DWELl
[SOURce<HW>]:SWEep:COMBined:MODE
[SOURce<HW>]:SWEep:COMBined:RETRace
[SOURce<HW>]:SWEep:COMBined:SHAPe
```

**class CombinedCls**

Combined commands group definition. 6 total commands, 1 Subgroups, 5 group commands

**get_count**() → int

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:COUNt
value: int = driver.source.sweep.combined.get_count()
```

Defines the number of sweeps you want to execute. This parameter applies to [:SOURce<hw>]:SWEep:COMBined:MODE > SINGle. To start the sweep signal generation, use the command [:SOURce<hw>]:SWEep:COMBined:EXECute.

**return**
step_count: integer Range: 1 to SeMAX_INT_STEP-1

**get_dwell**() → float

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:DWELl
value: float = driver.source.sweep.combined.get_dwell()
```

Sets the dwell time for the combined frequency / level sweep.

**return**
dwell: float Range: 0.01 to 100

**get_mode**() → AutoManStep

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:MODE
value: enums.AutoManStep = driver.source.sweep.combined.get_mode()
```

Sets the cycle mode for the combined frequency / level sweep.

**return**
sweep_comb_mode: AUTO| MANual| STEP AUTO Each trigger event triggers exactly one complete sweep. MANual The trigger system is not active. You can trigger every step individually by input of the frequencies with the commands [:SOURcehw]:FREQuency:MANual and [:SOURcehw]:POWer:MANual. STEP Each trigger event triggers one sweep step.

**get_retrace**() → bool

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:RETRace
value: bool = driver.source.sweep.combined.get_retrace()
```

Activates that the signal changes to the start level value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**return**
retrace_state: 1| ON| 0| OFF

**get_shape**() → SweCyclMode

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:SHAPe
value: enums.SweCyclMode = driver.source.sweep.combined.get_shape()
```

Selects the waveform shape for the combined frequency / level sweep sequence.

**return**
shape: SAWTooth| TRIangle

**set_count**(*step_count: int*) → None

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:COUNt
driver.source.sweep.combined.set_count(step_count = 1)
```

Defines the number of sweeps you want to execute. This parameter applies to [:SOURce<hw>]:SWEep:COMBined:MODE > SINGle. To start the sweep signal generation, use the command [:SOURce<hw>]:SWEep:COMBined:EXECute.

**param step_count**
integer Range: 1 to SeMAX_INT_STEP-1

**set_dwell**(*dwell: float*) → None

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:DWELl
driver.source.sweep.combined.set_dwell(dwell = 1.0)
```

Sets the dwell time for the combined frequency / level sweep.

**param dwell**
float Range: 0.01 to 100

**set_mode**(*sweep_comb_mode: AutoManStep*) → None

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:MODE
driver.source.sweep.combined.set_mode(sweep_comb_mode = enums.AutoManStep.AUTO)
```

Sets the cycle mode for the combined frequency / level sweep.

**param sweep_comb_mode**
AUTO| MANual| STEP AUTO Each trigger event triggers exactly one complete sweep. MANual The trigger system is not active. You can trigger every step individually by input of the frequencies with the commands [:SOURcehw]:FREQuency:MANual and [:SOURcehw]:POWer:MANual. STEP Each trigger event triggers one sweep step.

**set_retrace**(*retrace_state: bool*) → None

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:RETRace
driver.source.sweep.combined.set_retrace(retrace_state = False)
```

Activates that the signal changes to the start level value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**param retrace_state**
1| ON| 0| OFF

**set_shape**(*shape: SweCyclMode*) → None

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:SHAPe
driver.source.sweep.combined.set_shape(shape = enums.SweCyclMode.SAWTooth)
```

Selects the waveform shape for the combined frequency / level sweep sequence.

**param shape**
SAWTooth| TRIangle

### Cloning the Group

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

### Subgroups

#### 6.18.27.1.1 Execute

### SCPI Command :

```
[SOURce<HW>]:SWEep:COMBined:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:EXECute
driver.source.sweep.combined.execute.set()
```

Executes an RF frequency / level sweep cycle. The command triggers one single sweep manually. Therefore, you can use it in manual sweep mode, selected with the command [:SOURce<hw>]:SWEep:COMBined:MODE > MANual.

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

```
# SCPI: [SOURce<HW>]:SWEep:COMBined:EXECute
driver.source.sweep.combined.execute.set_with_opc()
```

Executes an RF frequency / level sweep cycle. The command triggers one single sweep manually. Therefore, you can use it in manual sweep mode, selected with the command [:SOURce<hw>]:SWEep:COMBined:MODE > MANual.

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

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

#### 6.18.27.2 Frequency

##### SCPI Commands :

```
[SOURce<HW>]:SWEep:[FREQuency]:DWELl
[SOURce<HW>]:SWEep:[FREQuency]:POINts
[SOURce<HW>]:SWEep:[FREQuency]:RETRace
[SOURce<HW>]:SWEep:[FREQuency]:RUNNing
[SOURce<HW>]:SWEep:[FREQuency]:SHAPe
[SOURce<HW>]:SWEep:[FREQuency]:SPACing
[SOURce<HW>]:SWEep:[FREQuency]:TIME
```

**class FrequencyCls**

Frequency commands group definition. 16 total commands, 5 Subgroups, 7 group commands

**get_dwell**() → float

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:DWELl
value: float = driver.source.sweep.frequency.get_dwell()
```

Sets the dwell time for a frequency sweep step.

**return**
dwell: float Range: 0.001 to 100

**get_points**() → int

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:POINts
value: int = driver.source.sweep.frequency.get_points()
```

Sets the number of steps within the RF frequency sweep range. See 'Correlating parameters in sweep mode'. Two separate POINts values are used for linear or logarithmic sweep spacing (LIN | LOG) . The command always affects the currently set sweep spacing.

**return**
points: integer Range: 2 to Max

**get_retrace**() → bool

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:RETRace
value: bool = driver.source.sweep.frequency.get_retrace()
```

Activates that the signal changes to the start frequency value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**return**
state: 1| ON| 0| OFF

**get_running**() → bool

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:RUNNing
value: bool = driver.source.sweep.frequency.get_running()
```

Queries the current sweep state.

**return**
state: 1| ON| 0| OFF

**get_shape**() → SweCyclMode

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:SHAPe
value: enums.SweCyclMode = driver.source.sweep.frequency.get_shape()
```

Determines the waveform shape for a frequency sweep sequence.

**return**
shape: SAWTooth| TRIangle

**get_spacing**() → Spacing

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:SPACing
value: enums.Spacing = driver.source.sweep.frequency.get_spacing()
```

Selects the mode for the calculation of the frequency intervals, with which the current frequency at each step is increased or decreased. The keyword [:FREQuency] can be omitted; then the command is SCPI-compliant.

**return**
spacing: LINear| LOGarithmic LINear Sets a fixed frequency value as step width and adds it to the current frequency. The linear step width is entered in Hz, see [:SOURcehw]:SWEep[:FREQuency]:STEP[:LINear]. LOGarithmic Sets a constant fraction of the current frequency as step width and adds it to the current frequency. The logarithmic step width is entered in %, see [:SOURcehw]:SWEep[:FREQuency]:STEP:LOGarithmic.

**get_time**() → float

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:TIME
value: float = driver.source.sweep.frequency.get_time()
```

Sets the duration of a frequency ramp sweep step.

**return**
time: float Range: 0.01 to 100, Unit: s

**set_dwell**(*dwell: float*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:DWELl
driver.source.sweep.frequency.set_dwell(dwell = 1.0)
```

Sets the dwell time for a frequency sweep step.

**param dwell**
float Range: 0.001 to 100

**set_points**(*points: int*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:POINts
driver.source.sweep.frequency.set_points(points = 1)
```

Sets the number of steps within the RF frequency sweep range. See 'Correlating parameters in sweep mode'. Two separate POINts values are used for linear or logarithmic sweep spacing (LIN | LOG) . The command always affects the currently set sweep spacing.

**param points**
integer Range: 2 to Max

**set_retrace**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:RETRace
driver.source.sweep.frequency.set_retrace(state = False)
```

Activates that the signal changes to the start frequency value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**param state**
1| ON| 0| OFF

**set_shape**(*shape: SweCyclMode*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:SHAPe
driver.source.sweep.frequency.set_shape(shape = enums.SweCyclMode.SAWTooth)
```

Determines the waveform shape for a frequency sweep sequence.

**param shape**
SAWTooth| TRIangle

**set_spacing**(*spacing: Spacing*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:SPACing
driver.source.sweep.frequency.set_spacing(spacing = enums.Spacing.LINear)
```

Selects the mode for the calculation of the frequency intervals, with which the current frequency at each step is increased or decreased. The keyword [:FREQuency] can be omitted; then the command is SCPI-compliant.

**param spacing**
LINear| LOGarithmic LINear Sets a fixed frequency value as step width and adds it to the current frequency. The linear step width is entered in Hz, see [:SOURcehw]:SWEep[:FREQuency]:STEP[:LINear]. LOGarithmic Sets a constant fraction of the current frequency as step width and adds it to the current frequency. The logarithmic step width is entered in %, see [:SOURcehw]:SWEep[:FREQuency]:STEP:LOGarithmic.

**set_time**(*time: float*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:TIME
driver.source.sweep.frequency.set_time(time = 1.0)
```

Sets the duration of a frequency ramp sweep step.

**param time**
float Range: 0.01 to 100, Unit: s

### Cloning the Group

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

### Subgroups

#### 6.18.27.2.1 Analog

### SCPI Command :

```
[SOURce<HW>]:SWEep:[FREQuency]:ANALog:SWPoints
```

**class AnalogCls**

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

**get_sw_points**() → List[float]

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:ANALog:SWPoints
value: List[float] = driver.source.sweep.frequency.analog.get_sw_points()
```

Queries blank points during the RF frequency sweep in ramp sweep mode. At certain switchover frequency points, the R&S SMA100B shortly blanks the RF signal to adjust the settings accordingly. This query returns all blanked frequency points within the entire frequency range, regardless of the set frequency sweep range.

**return**
sweep_ramp_blank_points: No help available

#### 6.18.27.2.2 Execute

### SCPI Command :

```
[SOURce<HW>]:SWEep:[FREQuency]:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:EXECute
driver.source.sweep.frequency.execute.set()
```

Executes an RF frequency sweep. The command performs a single sweep and is therefore only effective in manual sweep mode.

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

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:EXECute
driver.source.sweep.frequency.execute.set_with_opc()
```

Executes an RF frequency sweep. The command performs a single sweep and is therefore only effective in manual sweep mode.

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

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

#### 6.18.27.2.3 Marker<Marker>

##### RepCap Settings

```
# Range: Nr0 .. Nr31
rc = driver.source.sweep.frequency.marker.repcap_marker_get()
driver.source.sweep.frequency.marker.repcap_marker_set(repcap.Marker.Nr0)
```

##### SCPI Command :

```
[SOURce<HW>]:SWEep:[FREQuency]:MARKer:ACTive
```

**class MarkerCls**

Marker commands group definition. 3 total commands, 2 Subgroups, 1 group commands Repeated Capability: Marker, default value after init: Marker.Nr0

**get_active**() → SweMarkActive

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MARKer:ACTive
value: enums.SweMarkActive = driver.source.sweep.frequency.marker.get_active()
```

Defines the marker signal to be output with a higher voltage than all other markers.

**return**
active: NONE| M01| M02| M03| M04| M05| M06| M07| M08| M09| M10

**set_active**(*active: SweMarkActive*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MARKer:ACTive
driver.source.sweep.frequency.marker.set_active(active = enums.SweMarkActive.
↪M01)
```

Defines the marker signal to be output with a higher voltage than all other markers.

**param active**
NONE| M01| M02| M03| M04| M05| M06| M07| M08| M09| M10

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.sweep.frequency.marker.clone()
```

### Subgroups

#### 6.18.27.2.3.1 Frequency

### SCPI Command :

```
[SOURce<HW>]:SWEep:[FREQuency]:MARKer<CH>:FREQuency
```

**class FrequencyCls**

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

**get**(*marker=Marker.Default*) → float

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MARKer<CH>:FREQuency
value: float = driver.source.sweep.frequency.marker.frequency.get(marker =␣
↪repcap.Marker.Default)
```

Sets the frequency of the selected marker.

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
frequency: float

**set**(*frequency: float*, *marker=Marker.Default*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MARKer<CH>:FREQuency
driver.source.sweep.frequency.marker.frequency.set(frequency = 1.0, marker =␣
↪repcap.Marker.Default)
```

Sets the frequency of the selected marker.

**param frequency**
float

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.18.27.2.3.2 Fstate

**SCPI Command :**

```
[SOURce<HW>]:SWEep:[FREQuency]:MARKer<CH>:FSTate
```

**class FstateCls**

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

**get**(*marker=Marker.Default*) → bool

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MARKer<CH>:FSTate
value: bool = driver.source.sweep.frequency.marker.fstate.get(marker = repcap.
↪Marker.Default)
```

Activates the selected marker.

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

**return**
fstate: 1| ON| 0| OFF

**set**(*fstate: bool*, *marker=Marker.Default*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MARKer<CH>:FSTate
driver.source.sweep.frequency.marker.fstate.set(fstate = False, marker = repcap.
↪Marker.Default)
```

Activates the selected marker.

**param fstate**
1| ON| 0| OFF

**param marker**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Marker')

#### 6.18.27.2.4 Mode

**SCPI Commands :**

```
[SOURce<HW>]:SWEep:[FREQuency]:MODE:ADVanced
[SOURce<HW>]:SWEep:[FREQuency]:MODE
```

**class ModeCls**

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

**get_advanced**() → AutoManualMode

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MODE:ADVanced
value: enums.AutoManualMode = driver.source.sweep.frequency.mode.get_advanced()
```

No command help available

**return**
adv_freq_mode_sel: No help available

**get_value**() → AutoManStep

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MODE
value: enums.AutoManStep = driver.source.sweep.frequency.mode.get_value()
```

Sets the cycle mode for the frequency sweep.

**return**
mode: AUTO| MANual| STEP AUTO Each trigger event triggers exactly one complete sweep. MANual The trigger system is not active. You can trigger every step individually by input of the frequencies with the command [:SOURcehw]:FREQuency:MANual. STEP Each trigger event triggers one sweep step. The frequency increases by the value entered with [:SOURcehw]:SWEep[:FREQuency]:STEP[:LINear] (linear spacing) or [:SOURcehw]:SWEep[:FREQuency]:STEP:LOGarithmic (logarithmic spacing) .

**set_advanced**(*adv_freq_mode_sel: AutoManualMode*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MODE:ADVanced
driver.source.sweep.frequency.mode.set_advanced(adv_freq_mode_sel = enums.
↪AutoManualMode.AUTO)
```

No command help available

**param adv_freq_mode_sel**
No help available

**set_value**(*mode: AutoManStep*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MODE
driver.source.sweep.frequency.mode.set_value(mode = enums.AutoManStep.AUTO)
```

Sets the cycle mode for the frequency sweep.

**param mode**
AUTO| MANual| STEP AUTO Each trigger event triggers exactly one complete sweep. MANual The trigger system is not active. You can trigger every step individually by input of the frequencies with the command [:SOURcehw]:FREQuency:MANual. STEP Each trigger event triggers one sweep step. The frequency increases by the value entered with [:SOURcehw]:SWEep[:FREQuency]:STEP[:LINear] (linear spacing) or [:SOURcehw]:SWEep[:FREQuency]:STEP:LOGarithmic (logarithmic spacing) .

#### 6.18.27.2.5 Step

**SCPI Commands :**

```
[SOURce<HW>]:SWEep:[FREQuency]:STEP:LOGarithmic
[SOURce<HW>]:SWEep:[FREQuency]:STEP:[LINear]
```

**class StepCls**

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

**get_linear**() → float

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:STEP:[LINear]
value: float = driver.source.sweep.frequency.step.get_linear()
```

Sets the step width for linear sweeps. See 'Correlating parameters in sweep mode'. Omit the optional keywords so that the command is SCPI-compliant.

**return**
linear: float Range: 0.001 Hz to (STOP - STARt)

**get_logarithmic**() → float

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:STEP:LOGarithmic
value: float = driver.source.sweep.frequency.step.get_logarithmic()
```

Sets a logarithmically determined step width for the RF frequency sweep. The value is added at each sweep step to the current frequency. See 'Correlating parameters in sweep mode'.

**return**
logarithmic: float The unit is mandatory. Range: 0.01 to 100, Unit: PCT

**set_linear**(*linear: float*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:STEP:[LINear]
driver.source.sweep.frequency.step.set_linear(linear = 1.0)
```

Sets the step width for linear sweeps. See 'Correlating parameters in sweep mode'. Omit the optional keywords so that the command is SCPI-compliant.

**param linear**
float Range: 0.001 Hz to (STOP - STARt)

**set_logarithmic**(*logarithmic: float*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:STEP:LOGarithmic
driver.source.sweep.frequency.step.set_logarithmic(logarithmic = 1.0)
```

Sets a logarithmically determined step width for the RF frequency sweep. The value is added at each sweep step to the current frequency. See 'Correlating parameters in sweep mode'.

**param logarithmic**
float The unit is mandatory. Range: 0.01 to 100, Unit: PCT

#### 6.18.27.3 Marker

**class MarkerCls**

Marker 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.source.sweep.marker.clone()
```

#### Subgroups

##### 6.18.27.3.1 Output

#### SCPI Command :

```
[SOURce<HW>]:SWEep:MARKer:OUTPut:POLarity
```

**class OutputCls**

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

**get_polarity**() → NormalInverted

```
# SCPI: [SOURce<HW>]:SWEep:MARKer:OUTPut:POLarity
value: enums.NormalInverted = driver.source.sweep.marker.output.get_polarity()
```

Selects the polarity of the marker signal.

**return**
polarity: NORMal| INVerted NORMal Marker level is high when after reaching the mark. INVerted Marker level is low after reaching the mark.

**set_polarity**(*polarity: NormalInverted*) → None

```
# SCPI: [SOURce<HW>]:SWEep:MARKer:OUTPut:POLarity
driver.source.sweep.marker.output.set_polarity(polarity = enums.NormalInverted.
↪INVerted)
```

Selects the polarity of the marker signal.

**param polarity**
NORMal| INVerted NORMal Marker level is high when after reaching the mark. INVerted Marker level is low after reaching the mark.

#### 6.18.27.4 Power

#### SCPI Commands :

```
[SOURce<HW>]:SWEep:POWer:AMODe
[SOURce<HW>]:SWEep:POWer:DWELl
[SOURce<HW>]:SWEep:POWer:POINts
[SOURce<HW>]:SWEep:POWer:RETRace
[SOURce<HW>]:SWEep:POWer:RUNNing
[SOURce<HW>]:SWEep:POWer:SHAPe
```

**class PowerCls**

Power commands group definition. 11 total commands, 4 Subgroups, 6 group commands

**get_amode**() → PowAttMode

```
# SCPI: [SOURce<HW>]:SWEep:POWer:AMODe
value: enums.PowAttMode = driver.source.sweep.power.get_amode()
```

No command help available

**return**
amode: No help available

**get_dwell**() → float

```
# SCPI: [SOURce<HW>]:SWEep:POWer:DWELl
value: float = driver.source.sweep.power.get_dwell()
```

Sets the dwell time for a level sweep step.

**return**
dwell: float Range: 0.001 to 100

**get_points**() → int

```
# SCPI: [SOURce<HW>]:SWEep:POWer:POINts
value: int = driver.source.sweep.power.get_points()
```

Sets the number of steps within the RF level sweep range. See 'Correlating parameters in sweep mode'.

**return**
points: integer Range: 2 to Max

**get_retrace**() → bool

```
# SCPI: [SOURce<HW>]:SWEep:POWer:RETRace
value: bool = driver.source.sweep.power.get_retrace()
```

Activates that the signal changes to the start frequency value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**return**
state: 1| ON| 0| OFF

**get_running**() → bool

```
# SCPI: [SOURce<HW>]:SWEep:POWer:RUNNing
value: bool = driver.source.sweep.power.get_running()
```

Queries the current sweep state.

**return**
state: 1| ON| 0| OFF

**get_shape**() → SweCyclMode

```
# SCPI: [SOURce<HW>]:SWEep:POWer:SHAPe
value: enums.SweCyclMode = driver.source.sweep.power.get_shape()
```

Determines the waveform shape for a frequency sweep sequence.

**return**
shape: SAWTooth| TRIangle

**set_amode**(*amode: PowAttMode*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:AMODe
driver.source.sweep.power.set_amode(amode = enums.PowAttMode.AUTO)
```

No command help available

**param amode**
No help available

**set_dwell**(*dwell: float*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:DWELl
driver.source.sweep.power.set_dwell(dwell = 1.0)
```

Sets the dwell time for a level sweep step.

**param dwell**
float Range: 0.001 to 100

**set_points**(*points: int*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:POINts
driver.source.sweep.power.set_points(points = 1)
```

Sets the number of steps within the RF level sweep range. See 'Correlating parameters in sweep mode'.

**param points**
integer Range: 2 to Max

**set_retrace**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:RETRace
driver.source.sweep.power.set_retrace(state = False)
```

Activates that the signal changes to the start frequency value while it is waiting for the next trigger event. You can enable this feature, when you are working with sawtooth shapes in sweep mode 'Single' or 'External Single'.

**param state**
1| ON| 0| OFF

**set_shape**(*shape: SweCyclMode*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:SHAPe
driver.source.sweep.power.set_shape(shape = enums.SweCyclMode.SAWTooth)
```

Determines the waveform shape for a frequency sweep sequence.

**param shape**
SAWTooth| TRIangle

#### Cloning the Group

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

#### Subgroups

##### 6.18.27.4.1 Execute

#### SCPI Command :

```
[SOURce<HW>]:SWEep:POWer:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:EXECute
driver.source.sweep.power.execute.set()
```

Executes an RF frequency sweep. The command performs a single sweep and is therefore only effective in manual sweep mode.

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

```
# SCPI: [SOURce<HW>]:SWEep:POWer:EXECute
driver.source.sweep.power.execute.set_with_opc()
```

Executes an RF frequency sweep. The command performs a single sweep and is therefore only effective in manual sweep mode.

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

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

##### 6.18.27.4.2 Mode

#### SCPI Commands :

```
[SOURce<HW>]:SWEep:POWer:MODE:ADVanced
[SOURce<HW>]:SWEep:POWer:MODE
```

**class ModeCls**

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

**get_advanced**() → AutoManualMode

```
# SCPI: [SOURce<HW>]:SWEep:POWer:MODE:ADVanced
value: enums.AutoManualMode = driver.source.sweep.power.mode.get_advanced()
```

No command help available

**return**
pow_mode_adv: No help available

**get_value**() → AutoManStep

```
# SCPI: [SOURce<HW>]:SWEep:POWer:MODE
value: enums.AutoManStep = driver.source.sweep.power.mode.get_value()
```

Sets the cycle mode for the level sweep.

**return**
mode: AUTO| MANual| STEP AUTO Each trigger triggers exactly one complete sweep. MANual The trigger system is not active. You can trigger every step individually with the command [:SOURcehw]:POWer:MANual. The level value increases at each step by the value that you define with [:SOURcehw]:POWer:STEP[:INCRement]. Values directly entered with the command [:SOURcehw]:POWer:MANual are not taken into account. STEP Each trigger triggers one sweep step only. The level increases by the value entered with [:SOURcehw]:POWer:STEP[:INCRement].

**set_advanced**(*pow_mode_adv: AutoManualMode*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:MODE:ADVanced
driver.source.sweep.power.mode.set_advanced(pow_mode_adv = enums.AutoManualMode.
↪AUTO)
```

No command help available

**param pow_mode_adv**
No help available

**set_value**(*mode: AutoManStep*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:MODE
driver.source.sweep.power.mode.set_value(mode = enums.AutoManStep.AUTO)
```

Sets the cycle mode for the level sweep.

**param mode**
AUTO| MANual| STEP AUTO Each trigger triggers exactly one complete sweep. MANual The trigger system is not active. You can trigger every step individually with the command [:SOURcehw]:POWer:MANual. The level value increases at each step by the value that you define with [:SOURcehw]:POWer:STEP[:INCRement]. Values directly entered with the command [:SOURcehw]:POWer:MANual are not taken into account. STEP Each trigger triggers one sweep step only. The level increases by the value entered with [:SOURcehw]:POWer:STEP[:INCRement].

#### 6.18.27.4.3 Spacing

**SCPI Command :**

```
[SOURce<HW>]:SWEep:POWer:SPACing:MODE
```

**class SpacingCls**

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

**get_mode**() → Spacing

```
# SCPI: [SOURce<HW>]:SWEep:POWer:SPACing:MODE
value: enums.Spacing = driver.source.sweep.power.spacing.get_mode()
```

Queries the level sweep spacing. The sweep spacing for level sweeps is always linear.

**return**
mode: LINear

#### 6.18.27.4.4 Step

**SCPI Command :**

```
[SOURce<HW>]:SWEep:POWer:STEP:[LOGarithmic]
```

**class StepCls**

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

**get_logarithmic**() → float

```
# SCPI: [SOURce<HW>]:SWEep:POWer:STEP:[LOGarithmic]
value: float = driver.source.sweep.power.step.get_logarithmic()
```

Sets a logarithmically determined step size for the RF level sweep. The level is increased by a logarithmically calculated fraction of the current level. See 'Correlating parameters in sweep mode'.

**return**
logarithmic: float The unit dB is mandatory. Range: 0.01 to 139 dB, Unit: dB

**set_logarithmic**(*logarithmic: float*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:STEP:[LOGarithmic]
driver.source.sweep.power.step.set_logarithmic(logarithmic = 1.0)
```

Sets a logarithmically determined step size for the RF level sweep. The level is increased by a logarithmically calculated fraction of the current level. See 'Correlating parameters in sweep mode'.

**param logarithmic**
float The unit dB is mandatory. Range: 0.01 to 139 dB, Unit: dB

### 6.18.28 ValRf

#### SCPI Command :

```
[SOURce<HW>]:VALRf:SLOPe
```

**class ValRfCls**

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

**get_slope**() → SlopeType

```
# SCPI: [SOURce<HW>]:VALRf:SLOPe
value: enums.SlopeType = driver.source.valRf.get_slope()
```

No command help available

**return**
sig_valid_slope: No help available

**set_slope**(*sig_valid_slope: SlopeType*) → None

```
# SCPI: [SOURce<HW>]:VALRf:SLOPe
driver.source.valRf.set_slope(sig_valid_slope = enums.SlopeType.NEGative)
```

No command help available

**param sig_valid_slope**
No help available

### 6.18.29 Vor

#### SCPI Commands :

```
[SOURce<HW>]:VOR:MODE
[SOURce<HW>]:VOR:PRESet
[SOURce<HW>]:VOR:SOURce
[SOURce<HW>]:VOR:STATe
```

**class VorCls**

Vor commands group definition. 31 total commands, 8 Subgroups, 4 group commands

**get_mode**() → AvionicVorMode

```
# SCPI: [SOURce<HW>]:VOR:MODE
value: enums.AvionicVorMode = driver.source.vor.get_mode()
```

Sets the operating mode for the VOR modulation signal.

**return**
mode: NORM| VAR| SUBCarrier| FMSubcarrier NORM VOR modulation is active. VAR Amplitude modulation of the output signal with the variable signal component (30Hz signal content) of the VOR signal. The modulation depth of the 30 Hz signal can be set with [:SOURcehw]:VOR:VAR[:DEPTh]. SUBCarrier Amplitude modulation of the output signal with the unmodulated FM carrier (9960Hz) of the VOR signal. The modulation depth of the 30 Hz signal can be

set with [:SOURcehw]:VOR:SUBCarrier:DEPTh. FMSubcarrier Amplitude modulation of the output signal with the frequency modulated FM carrier (9960Hz) of the VOR signal. The modulation depth of the 30 Hz signal can be set with [:SOURcehw]:VOR:SUBCarrier:DEPTh. The frequency deviation can be set with [:SOURcehw]:VOR:REFerence[:DEViation].

**get_source**() → AvionicExtAm

```
# SCPI: [SOURce<HW>]:VOR:SOURce
value: enums.AvionicExtAm = driver.source.vor.get_source()
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**return**
vor_source_sel: INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:VOR:STATe
value: bool = driver.source.vor.get_state()
```

Activates/deactivates the VOR modulation.

**return**
state: 1| ON| 0| OFF

**preset**() → None

```
# SCPI: [SOURce<HW>]:VOR:PRESet
driver.source.vor.preset()
```

Sets the parameters of the digital standard to their default values (*RST values specified for the commands) . Not affected is the state set with the command SOURce<hw>:VOR:STATe.

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

```
# SCPI: [SOURce<HW>]:VOR:PRESet
driver.source.vor.preset_with_opc()
```

Sets the parameters of the digital standard to their default values (*RST values specified for the commands) . Not affected is the state set with the command SOURce<hw>:VOR:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_mode**(*mode: AvionicVorMode*) → None

```
# SCPI: [SOURce<HW>]:VOR:MODE
driver.source.vor.set_mode(mode = enums.AvionicVorMode.FMSubcarrier)
```

Sets the operating mode for the VOR modulation signal.

**param mode**
NORM| VAR| SUBCarrier| FMSubcarrier NORM VOR modulation is active. VAR Amplitude modulation of the output signal with the variable signal component (30Hz signal content) of the VOR signal. The modulation depth of the 30 Hz signal can be set with [:SOURcehw]:VOR:VAR[:DEPTh]. SUBCarrier Amplitude modulation of the output signal with the unmodulated FM carrier (9960Hz) of the VOR signal. The modulation depth of the 30 Hz signal can be set with [:SOURcehw]:VOR:SUBCarrier:DEPTh. FMSubcarrier Amplitude modulation of the output signal with the frequency modulated FM carrier (9960Hz) of the VOR signal. The modulation depth of the 30 Hz signal can be set with [:SOURcehw]:VOR:SUBCarrier:DEPTh. The frequency deviation can be set with [:SOURcehw]:VOR:REFerence[:DEViation].

**set_source**(*vor_source_sel: AvionicExtAm*) → None

```
# SCPI: [SOURce<HW>]:VOR:SOURce
driver.source.vor.set_source(vor_source_sel = enums.AvionicExtAm.EXT)
```

Sets the modulation source for the avionic standard modulation. If external modulation source is set, the external signal is added to the internal signal. Switching off the internal modulation source is not possible.

**param vor_source_sel**
INT| EXT| INT,EXT INT Internal modulation source is used. EXT|INT,EXT An external modulation source is used, additional to the internal modulation source. The external signal is input at the Ext connector.

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:VOR:STATe
driver.source.vor.set_state(state = False)
```

Activates/deactivates the VOR modulation.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.29.1 Bangle

### SCPI Commands :

```
[SOURce<HW>]:VOR:[BANGle]:DIRection
[SOURce<HW>]:VOR:[BANGle]
```

**class BangleCls**
Bangle commands group definition. 2 total commands, 0 Subgroups, 2 group commands

**get_direction**() → AvionicVorDir

```
# SCPI: [SOURce<HW>]:VOR:[BANGle]:DIRection
value: enums.AvionicVorDir = driver.source.vor.bangle.get_direction()
```

Sets the reference position of the phase information.

**return**
direction: FROM| TO FROM The bearing angle is measured between the geographic north and the connection line from beacon to airplane. TO The bearing angle is measured between the geographic north and the connection line from airplane to beacon.

**get_value**() → float

```
# SCPI: [SOURce<HW>]:VOR:[BANGle]
value: float = driver.source.vor.bangle.get_value()
```

Sets the bearing angle between the VAR signal and the reference signal. The orientation of the angle can be set with [:SOURce<hw>]:VOR[:BANGle]:DIRection.

**return**
bangle: float Range: 0 to 360

**set_direction**(*direction: AvionicVorDir*) → None

```
# SCPI: [SOURce<HW>]:VOR:[BANGle]:DIRection
driver.source.vor.bangle.set_direction(direction = enums.AvionicVorDir.FROM)
```

Sets the reference position of the phase information.

**param direction**
FROM| TO FROM The bearing angle is measured between the geographic north and the connection line from beacon to airplane. TO The bearing angle is measured between the geographic north and the connection line from airplane to beacon.

**set_value**(*bangle: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:[BANGle]
driver.source.vor.bangle.set_value(bangle = 1.0)
```

Sets the bearing angle between the VAR signal and the reference signal. The orientation of the angle can be set with [:SOURce<hw>]:VOR[:BANGle]:DIRection.

**param bangle**
float Range: 0 to 360

#### 6.18.29.2 Comid

**SCPI Commands :**

```
[SOURce<HW>]:VOR:COMid:DASH
[SOURce<HW>]:VOR:COMid:DEPTh
[SOURce<HW>]:VOR:COMid:DOT
[SOURce<HW>]:VOR:COMid:FREQuency
[SOURce<HW>]:VOR:COMid:LETTer
```

(continues on next page)

(continued from previous page)

```
[SOURce<HW>]:VOR:COMid:PERiod
[SOURce<HW>]:VOR:COMid:REPeat
[SOURce<HW>]:VOR:COMid:SYMBol
[SOURce<HW>]:VOR:COMid:TSCHema
[SOURce<HW>]:VOR:COMid:[STATe]
```

**class ComidCls**

Comid commands group definition. 12 total commands, 1 Subgroups, 10 group commands

**get_dash**() → float

```
# SCPI: [SOURce<HW>]:VOR:COMid:DASH
value: float = driver.source.vor.comid.get_dash()
```

Sets the length of a Morse code dash.

**return**
dash: float Range: 0.05 to 1

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:VOR:COMid:DEPTh
value: float = driver.source.vor.comid.get_depth()
```

Sets the AM modulation depth of the COM/ID signal.

**return**
depth: float Range: 0 to 100

**get_dot**() → float

```
# SCPI: [SOURce<HW>]:VOR:COMid:DOT
value: float = driver.source.vor.comid.get_dot()
```

Sets the length of a Morse code dot. If the time schema is set to standard, the dash length (= 3 times dot length) , symbol space (= dot length) and letter space (= 3 times dot length) is also determined by this entry.

**return**
dot: float Range: 0.05 to 1

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:VOR:COMid:FREQuency
value: float = driver.source.vor.comid.get_frequency()
```

Sets the frequency of the COM/ID signal.

**return**
frequency: float Range: 0.1 to 20E3

**get_letter**() → float

```
# SCPI: [SOURce<HW>]:VOR:COMid:LETTer
value: float = driver.source.vor.comid.get_letter()
```

Sets the length of a Morse code letter space.

**return**
letter: float Range: 0.05 to 1, Unit: s

**get_period**() → float

```
# SCPI: [SOURce<HW>]:VOR:COMid:PERiod
value: float = driver.source.vor.comid.get_period()
```

Sets the period of the COM/ID signal.

**return**
period: float Range: 0 to 120

**get_repeat**() → int

```
# SCPI: [SOURce<HW>]:VOR:COMid:REPeat
value: int = driver.source.vor.comid.get_repeat()
```

No command help available

**return**
repeat: No help available

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:VOR:COMid:[STATe]
value: bool = driver.source.vor.comid.get_state()
```

Enables/disables the COM/ID signal.

**return**
state: 1| ON| 0| OFF

**get_symbol**() → float

```
# SCPI: [SOURce<HW>]:VOR:COMid:SYMBol
value: float = driver.source.vor.comid.get_symbol()
```

Sets the length of the Morse code symbol space.

**return**
symbol: float Range: 0.05 to 1

**get_tschema**() → AvionicComIdTimeSchem

```
# SCPI: [SOURce<HW>]:VOR:COMid:TSCHema
value: enums.AvionicComIdTimeSchem = driver.source.vor.comid.get_tschema()
```

Sets the time schema of the Morse code for the COM/ID signal.

**return**
tschema: STD| USER STD Activates the standard time schema of the Morse code. The set dot length determines the dash length, which is 3 times the dot length. USER Activates the user-defined time schema of the Morse code. Dot and dash length, as well as symbol and letter space can be set separately.

**set_dash**(*dash: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:DASH
driver.source.vor.comid.set_dash(dash = 1.0)
```

Sets the length of a Morse code dash.

**param dash**
float Range: 0.05 to 1

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:DEPTh
driver.source.vor.comid.set_depth(depth = 1.0)
```

Sets the AM modulation depth of the COM/ID signal.

**param depth**
float Range: 0 to 100

**set_dot**(*dot: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:DOT
driver.source.vor.comid.set_dot(dot = 1.0)
```

Sets the length of a Morse code dot. If the time schema is set to standard, the dash length (= 3 times dot length) , symbol space (= dot length) and letter space (= 3 times dot length) is also determined by this entry.

**param dot**
float Range: 0.05 to 1

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:FREQuency
driver.source.vor.comid.set_frequency(frequency = 1.0)
```

Sets the frequency of the COM/ID signal.

**param frequency**
float Range: 0.1 to 20E3

**set_letter**(*letter: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:LETTer
driver.source.vor.comid.set_letter(letter = 1.0)
```

Sets the length of a Morse code letter space.

**param letter**
float Range: 0.05 to 1, Unit: s

**set_period**(*period: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:PERiod
driver.source.vor.comid.set_period(period = 1.0)
```

Sets the period of the COM/ID signal.

**param period**
float Range: 0 to 120

**set_repeat**(*repeat: int*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:REPeat
driver.source.vor.comid.set_repeat(repeat = 1)
```

No command help available

**param repeat**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:[STATe]
driver.source.vor.comid.set_state(state = False)
```

Enables/disables the COM/ID signal.

**param state**
1| ON| 0| OFF

**set_symbol**(*symbol: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:SYMBol
driver.source.vor.comid.set_symbol(symbol = 1.0)
```

Sets the length of the Morse code symbol space.

**param symbol**
float Range: 0.05 to 1

**set_tschema**(*tschema: AvionicComIdTimeSchem*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:TSCHema
driver.source.vor.comid.set_tschema(tschema = enums.AvionicComIdTimeSchem.STD)
```

Sets the time schema of the Morse code for the COM/ID signal.

**param tschema**
STD| USER STD Activates the standard time schema of the Morse code. The set dot length determines the dash length, which is 3 times the dot length. USER Activates the user-defined time schema of the Morse code. Dot and dash length, as well as symbol and letter space can be set separately.

### Cloning the Group

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

### Subgroups

#### 6.18.29.2.1 Code

### SCPI Commands :

```
[SOURce<HW>]:VOR:COMid:CODE:STATe
[SOURce<HW>]:VOR:COMid:CODE
```

**class CodeCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:VOR:COMid:CODE:STATe
value: bool = driver.source.vor.comid.code.get_state()
```

No command help available

**return**
state: No help available

**get_value**() → str

```
# SCPI: [SOURce<HW>]:VOR:COMid:CODE
value: str = driver.source.vor.comid.code.get_value()
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**return**
code: string

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:CODE:STATe
driver.source.vor.comid.code.set_state(state = False)
```

No command help available

**param state**
No help available

**set_value**(*code: str*) → None

```
# SCPI: [SOURce<HW>]:VOR:COMid:CODE
driver.source.vor.comid.code.set_value(code = 'abc')
```

Sets the coding of the COM/ID signal by the international short name of the airport (e.g. MUC for the Munich airport) . The COM/ID tone is sent according to the selected code, see 'Morse code settings'. If no coding is set, the COM/ID tone is sent uncoded (key down) .

**param code**
string

#### 6.18.29.3 Frequency

**SCPI Commands :**

```
[SOURce<HW>]:VOR:FREQuency:MODE
[SOURce<HW>]:VOR:FREQuency:STEP
[SOURce<HW>]:VOR:FREQuency
```

**class FrequencyCls**

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

**get_mode**() → AvionicCarrFreqMode

```
# SCPI: [SOURce<HW>]:VOR:FREQuency:MODE
value: enums.AvionicCarrFreqMode = driver.source.vor.frequency.get_mode()
```

Sets the mode for the carrier frequency of the signal.

**return**
mode: DECimal| ICAO DECimal Activates user-defined variation of the carrier frequency. ICAO Activates variation in predefined steps according to standard VOR transmitting frequencies (see Table 'VOR ICAO channels and frequencies (MHz) ') .

**get_step**() → AvionicKnobStep

```
# SCPI: [SOURce<HW>]:VOR:FREQuency:STEP
value: enums.AvionicKnobStep = driver.source.vor.frequency.get_step()
```

No command help available

**return**
step: No help available

**get_value**() → float

```
# SCPI: [SOURce<HW>]:VOR:FREQuency
value: float = driver.source.vor.frequency.get_value()
```

Sets the carrier frequency of the signal.

**return**
carrier_freq: float Range: 100E3 to 6E9

**set_mode**(*mode: AvionicCarrFreqMode*) → None

```
# SCPI: [SOURce<HW>]:VOR:FREQuency:MODE
driver.source.vor.frequency.set_mode(mode = enums.AvionicCarrFreqMode.DECimal)
```

Sets the mode for the carrier frequency of the signal.

**param mode**
DECimal| ICAO DECimal Activates user-defined variation of the carrier frequency. ICAO Activates variation in predefined steps according to standard VOR transmitting frequencies (see Table 'VOR ICAO channels and frequencies (MHz) ') .

**set_step**(*step: AvionicKnobStep*) → None

```
# SCPI: [SOURce<HW>]:VOR:FREQuency:STEP
driver.source.vor.frequency.set_step(step = enums.AvionicKnobStep.DECimal)
```

No command help available

**param step**
No help available

**set_value**(*carrier_freq: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:FREQuency
driver.source.vor.frequency.set_value(carrier_freq = 1.0)
```

Sets the carrier frequency of the signal.

**param carrier_freq**
float Range: 100E3 to 6E9

#### 6.18.29.4 Icao

**SCPI Command :**

```
[SOURce<HW>]:VOR:ICAO:CHANnel
```

**class IcaoCls**

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

**get_channel**() → AvionicVorIcaoChan

```
# SCPI: [SOURce<HW>]:VOR:ICAO:CHANnel
value: enums.AvionicVorIcaoChan = driver.source.vor.icao.get_channel()
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The carrier frequency is set automatically to the value of the ICAO channel. For an overview of the VOR ICAO channel frequencies, see Table 'VOR ICAO channels and frequencies (MHz) '.

**return**
channel: CH17X| CH17Y| CH19X| CH19Y| CH21X| CH21Y| CH23X| CH23Y| CH25X| CH25Y| CH27X| CH27Y| CH29X| CH29Y| CH31X| CH31Y| CH33X| CH33Y| CH35X| CH35Y| CH37X| CH37Y| CH39X| CH39Y| CH41X| CH41Y| CH43X| CH43Y| CH45X| CH45Y| CH47X| CH47Y| CH49X| CH49Y| CH51X| CH51Y| CH53X| CH53Y| CH55X| CH55Y| CH57X| CH57Y| CH58X| CH58Y| CH59X| CH59Y| CH70X| CH70Y| CH71X| CH71Y| CH72X| CH72Y| CH73X| CH73Y| CH74X| CH74Y| CH75X| CH75Y| CH76X| CH76Y| CH77X| CH77Y| CH78X| CH78Y| CH79X| CH79Y| CH80X| CH80Y| CH81X| CH81Y| CH82X| CH82Y| CH83X| CH83Y| CH84X| CH84Y| CH85X| CH85Y| CH86X| CH86Y| CH87X| CH87Y| CH88X| CH88Y| CH89X| CH89Y| CH90X| CH90Y| CH91X| CH91Y| CH92X| CH92Y| CH93X| CH93Y| CH94X| CH94Y| CH95X| CH95Y| CH96X| CH96Y| CH97X| CH97Y| CH98X| CH98Y| CH99X| CH99Y| CH100X| CH100Y| CH101X| CH101Y| CH102X| CH102Y| CH103X| CH103Y| CH104X| CH104Y| CH105X| CH105Y| CH106X| CH106Y| CH107X| CH107Y| CH108X|

CH108Y| CH109X| CH109Y| CH110X| CH110Y| CH111X| CH111Y| CH112X| CH112Y| CH113X| CH113Y| CH114X| CH114Y| CH115X| CH115Y| CH116X| CH116Y| CH117X| CH117Y| CH118X| CH118Y| CH119X| CH119Y| CH120X| CH120Y| CH121X| CH121Y| CH122X| CH122Y| CH123X| CH123Y| CH124X| CH124Y| CH125X| CH125Y| CH126X| CH126Y

**set_channel**(*channel: AvionicVorIcaoChan*) → None

```
# SCPI: [SOURce<HW>]:VOR:ICAO:CHANnel
driver.source.vor.icao.set_channel(channel = enums.AvionicVorIcaoChan.CH100X)
```

Sets the ICAO channel and the corresponding transmitting frequency. If avionic standard modulation is activated and you change the 'RF Frequency', the frequency value of the closest ICAO channel is applied automatically. The 'ICAO Channel' is also updated. The carrier frequency is set automatically to the value of the ICAO channel. For an overview of the VOR ICAO channel frequencies, see Table 'VOR ICAO channels and frequencies (MHz) '.

**param channel**
CH17X| CH17Y| CH19X| CH19Y| CH21X| CH21Y| CH23X| CH23Y| CH25X| CH25Y| CH27X| CH27Y| CH29X| CH29Y| CH31X| CH31Y| CH33X| CH33Y| CH35X| CH35Y| CH37X| CH37Y| CH39X| CH39Y| CH41X| CH41Y| CH43X| CH43Y| CH45X| CH45Y| CH47X| CH47Y| CH49X| CH49Y| CH51X| CH51Y| CH53X| CH53Y| CH55X| CH55Y| CH57X| CH57Y| CH58X| CH58Y| CH59X| CH59Y| CH70X| CH70Y| CH71X| CH71Y| CH72X| CH72Y| CH73X| CH73Y| CH74X| CH74Y| CH75X| CH75Y| CH76X| CH76Y| CH77X| CH77Y| CH78X| CH78Y| CH79X| CH79Y| CH80X| CH80Y| CH81X| CH81Y| CH82X| CH82Y| CH83X| CH83Y| CH84X| CH84Y| CH85X| CH85Y| CH86X| CH86Y| CH87X| CH87Y| CH88X| CH88Y| CH89X| CH89Y| CH90X| CH90Y| CH91X| CH91Y| CH92X| CH92Y| CH93X| CH93Y| CH94X| CH94Y| CH95X| CH95Y| CH96X| CH96Y| CH97X| CH97Y| CH98X| CH98Y| CH99X| CH99Y| CH100X| CH100Y| CH101X| CH101Y| CH102X| CH102Y| CH103X| CH103Y| CH104X| CH104Y| CH105X| CH105Y| CH106X| CH106Y| CH107X| CH107Y| CH108X| CH108Y| CH109X| CH109Y| CH110X| CH110Y| CH111X| CH111Y| CH112X| CH112Y| CH113X| CH113Y| CH114X| CH114Y| CH115X| CH115Y| CH116X| CH116Y| CH117X| CH117Y| CH118X| CH118Y| CH119X| CH119Y| CH120X| CH120Y| CH121X| CH121Y| CH122X| CH122Y| CH123X| CH123Y| CH124X| CH124Y| CH125X| CH125Y| CH126X| CH126Y

#### 6.18.29.5 Reference

**SCPI Command :**

```
[SOURce<HW>]:VOR:REFerence:[DEViation]
```

**class ReferenceCls**

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

**get_deviation**() → int

```
# SCPI: [SOURce<HW>]:VOR:REFerence:[DEViation]
value: int = driver.source.vor.reference.get_deviation()
```

Sets the frequency deviation of the reference signal on the FM carrier.

**return**
deviation: integer Range: 0 to 960

**set_deviation**(*deviation: int*) → None

```
# SCPI: [SOURce<HW>]:VOR:REFerence:[DEViation]
driver.source.vor.reference.set_deviation(deviation = 1)
```

Sets the frequency deviation of the reference signal on the FM carrier.

**param deviation**
integer Range: 0 to 960

#### 6.18.29.6 Setting

**SCPI Commands :**

```
[SOURce<HW>]:VOR:SETTing:CATalog
[SOURce<HW>]:VOR:SETTing:DELete
[SOURce<HW>]:VOR:SETTing:LOAD
[SOURce<HW>]:VOR:SETTing:STORe
```

**class SettingCls**

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

**delete**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:VOR:SETTing:DELete
driver.source.vor.setting.delete(filename = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path; file extension can be omitted

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:VOR:SETTing:CATalog
value: List[str] = driver.source.vor.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**return**
avionic_vor_cat_names: filename1,filename2,... Returns a string of filenames separated by commas.

**load**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:VOR:SETTing:LOAD
driver.source.vor.setting.load(filename = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.adf/*.ils/*.vor. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path; file extension can be omitted

**set_store**(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:VOR:SETTing:STORe
driver.source.vor.setting.set_store(filename = 'abc')
```

Saves the current settings into the selected file; the file extension (*.adf/*.ils/*.vor) is assigned automatically. Refer to 'Accessing files in the default or in a specified directory' for general information on file handling in the default and in a specific directory.

**param filename**
'filename' Filename or complete file path

#### 6.18.29.7 Subcarrier

**SCPI Commands :**

```
[SOURce<HW>]:VOR:SUBCarrier:DEPTh
[SOURce<HW>]:VOR:SUBCarrier:[FREQuency]
```

**class SubcarrierCls**

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

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:VOR:SUBCarrier:DEPTh
value: float = driver.source.vor.subcarrier.get_depth()
```

Sets the AM modulation depth of the FM carrier.

**return**
depth: float Range: 0 to 100

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:VOR:SUBCarrier:[FREQuency]
value: float = driver.source.vor.subcarrier.get_frequency()
```

Sets the frequency of the FM carrier.

**return**
frequency: float Range: 5E3 to 15E3

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:SUBCarrier:DEPTh
driver.source.vor.subcarrier.set_depth(depth = 1.0)
```

Sets the AM modulation depth of the FM carrier.

**param depth**
float Range: 0 to 100

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:SUBCarrier:[FREQuency]
driver.source.vor.subcarrier.set_frequency(frequency = 1.0)
```

Sets the frequency of the FM carrier.

**param frequency**
float Range: 5E3 to 15E3

#### 6.18.29.8 Var

**SCPI Commands :**

```
[SOURce<HW>]:VOR:VAR:FREQuency
[SOURce<HW>]:VOR:VAR:[DEPTh]
```

**class VarCls**

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

**get_depth**() → float

```
# SCPI: [SOURce<HW>]:VOR:VAR:[DEPTh]
value: float = driver.source.vor.var.get_depth()
```

Sets the AM modulation depth of the 30Hz variable signal.

**return**
depth: float Range: 0 to 100

**get_frequency**() → float

```
# SCPI: [SOURce<HW>]:VOR:VAR:FREQuency
value: float = driver.source.vor.var.get_frequency()
```

Sets the frequency of the variable and the reference signal. As the two signals must have the same frequency, the setting is valid for both signals.

**return**
frequency: float Range: 10 to 60

**set_depth**(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:VAR:[DEPTh]
driver.source.vor.var.set_depth(depth = 1.0)
```

Sets the AM modulation depth of the 30Hz variable signal.

**param depth**
float Range: 0 to 100

**set_frequency**(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:VOR:VAR:FREQuency
driver.source.vor.var.set_frequency(frequency = 1.0)
```

Sets the frequency of the variable and the reference signal. As the two signals must have the same frequency, the setting is valid for both signals.

**param frequency**
float Range: 10 to 60

## 6.19 Status

### SCPI Command :

```
STATus:PRESet
```

**class StatusCls**

Status commands group definition. 22 total commands, 3 Subgroups, 1 group commands

**get_preset**() → str

```
# SCPI: STATus:PRESet
value: str = driver.status.get_preset()
```

Resets the status registers. All PTRansition parts are set to FFFFh (32767) , i.e. all transitions from 0 to 1 are detected. All NTRansition parts are set to 0, i.e. a transition from 1 to 0 in a CONDition bit is not detected. The ENABle parts of STATus:OPERation and STATus:QUEStionable are set to 0, i.e. all events in these registers are not passed on.

**return**
preset: string

**set_preset**(*preset: str*) → None

```
# SCPI: STATus:PRESet
driver.status.set_preset(preset = 'abc')
```

Resets the status registers. All PTRansition parts are set to FFFFh (32767) , i.e. all transitions from 0 to 1 are detected. All NTRansition parts are set to 0, i.e. a transition from 1 to 0 in a CONDition bit is not detected. The ENABle parts of STATus:OPERation and STATus:QUEStionable are set to 0, i.e. all events in these registers are not passed on.

**param preset**
string

#### Cloning the Group

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

#### Subgroups

### 6.19.1 Operation

#### SCPI Commands :

```
STATus:OPERation:CONDition
STATus:OPERation:ENABle
STATus:OPERation:NTRansition
STATus:OPERation:PTRansition
STATus:OPERation:[EVENt]
```

**class OperationCls**

Operation commands group definition. 10 total commands, 1 Subgroups, 5 group commands

**get_condition**() → str

```
# SCPI: STATus:OPERation:CONDition
value: str = driver.status.operation.get_condition()
```

Quieries 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 hardware status.

**return**
condition: string

**get_enable**() → str

```
# SCPI: STATus:OPERation:ENABle
value: str = driver.status.operation.get_enable()
```

Sets the bits of the ENABle part of the STATus:OPERation register. This setting determines which events of the Status-Event part are forwarded to the sum bit in the status byte. These events can be used for a service request.

**return**
enable: string

**get_event**() → str

```
# SCPI: STATus:OPERation:[EVENt]
value: str = driver.status.operation.get_event()
```

Queries the content of the EVENt part of the STATus:OPERation register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**return**
value: No help available

**get_ntransition**() → str

```
# SCPI: STATus:OPERation:NTRansition
value: str = driver.status.operation.get_ntransition()
```

Sets the bits of the NTRansition part of the STATus:OPERation register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register. The disappearance of an event in the hardware is thus registered, for example the end of an adjustment.

**return**
ntransition: string

**get_ptransition**() → str

```
# SCPI: STATus:OPERation:PTRansition
value: str = driver.status.operation.get_ptransition()
```

Sets the bits of the PTRansition part of the STATus:OPERation register. If a bit is set, a transition from 0 to 1 in the condition part causes an entry to be made in the EVENt part of the register. A new event in the hardware is thus registered, for example the start of an adjustment.

**return**
ptransition: string

**set_enable**(*enable: str*) → None

```
# SCPI: STATus:OPERation:ENABle
driver.status.operation.set_enable(enable = 'abc')
```

Sets the bits of the ENABle part of the STATus:OPERation register. This setting determines which events of the Status-Event part are forwarded to the sum bit in the status byte. These events can be used for a service request.

**param enable**
string

**set_event**(*value: str*) → None

```
# SCPI: STATus:OPERation:[EVENt]
driver.status.operation.set_event(value = 'abc')
```

Queries the content of the EVENt part of the STATus:OPERation register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**param value**
string

**set_ntransition**(*ntransition: str*) → None

```
# SCPI: STATus:OPERation:NTRansition
driver.status.operation.set_ntransition(ntransition = 'abc')
```

Sets the bits of the NTRansition part of the STATus:OPERation register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register. The disappearance of an event in the hardware is thus registered, for example the end of an adjustment.

**param ntransition**
string

**set_ptransition**(*ptransition: str*) → None

```
# SCPI: STATus:OPERation:PTRansition
driver.status.operation.set_ptransition(ptransition = 'abc')
```

Sets the bits of the PTRansition part of the STATus:OPERation register. If a bit is set, a transition from 0 to 1 in the condition part causes an entry to be made in the EVENt part of the register. A new event in the hardware is thus registered, for example the start of an adjustment.

**param ptransition**
string

### Cloning the Group

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

### Subgroups

#### 6.19.1.1 Bit<BitNumberNull>

### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.status.operation.bit.repcap_bitNumberNull_get()
driver.status.operation.bit.repcap_bitNumberNull_set(repcap.BitNumberNull.Nr0)
```

**class BitCls**

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: BitNumberNull, default value after init: BitNumberNull.Nr0

### Cloning the Group

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

### Subgroups

#### 6.19.1.1.1 Condition

### SCPI Command :

```
STATus:OPERation:BIT<BITNR>:CONDition
```

**class ConditionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:CONDition
value: str = driver.status.operation.bit.condition.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
condition: No help available

#### 6.19.1.1.2 Enable

##### SCPI Command :

```
STATus:OPERation:BIT<BITNR>:ENABle
```

**class EnableCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:ENABle
value: str = driver.status.operation.bit.enable.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
enable: No help available

**set**(*enable: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:OPERation:BIT<BITNR>:ENABle
driver.status.operation.bit.enable.set(enable = 'abc', bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param enable**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.19.1.1.3 Event

**SCPI Command :**

```
STATus:OPERation:BIT<BITNR>:[EVENt]
```

**class EventCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:[EVENt]
value: str = driver.status.operation.bit.event.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
event: No help available

#### 6.19.1.1.4 Ntransition

**SCPI Command :**

```
STATus:OPERation:BIT<BITNR>:NTRansition
```

**class NtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:NTRansition
value: str = driver.status.operation.bit.ntransition.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ntransition: No help available

**set**(*ntransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:OPERation:BIT<BITNR>:NTRansition
driver.status.operation.bit.ntransition.set(ntransition = 'abc', bitNumberNull␣
↪= repcap.BitNumberNull.Default)
```

No command help available

**param ntransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.19.1.1.5 Ptransition

**SCPI Command :**

```
STATus:OPERation:BIT<BITNR>:PTRansition
```

**class PtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:PTRansition
value: str = driver.status.operation.bit.ptransition.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ptransition: No help available

**set**(*ptransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:OPERation:BIT<BITNR>:PTRansition
driver.status.operation.bit.ptransition.set(ptransition = 'abc', bitNumberNull␣
↪= repcap.BitNumberNull.Default)
```

No command help available

**param ptransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

### 6.19.2 Questionable

**SCPI Commands :**

```
STATus:QUEStionable:CONDition
STATus:QUEStionable:ENABle
STATus:QUEStionable:NTRansition
STATus:QUEStionable:PTRansition
STATus:QUEStionable:[EVENt]
```

**class QuestionableCls**

Questionable commands group definition. 10 total commands, 1 Subgroups, 5 group commands

**get_condition**() → str

```
# SCPI: STATus:QUEStionable:CONDition
value: str = driver.status.questionable.get_condition()
```

Queries the content of the CONDition part of the STATus:QUEStionable register. This part contains information on the action currently being performed in the instrument. The content is not deleted after being read out since it indicates the current hardware status.

**return**
condition: string

**get_enable**() → str

```
# SCPI: STATus:QUEStionable:ENABle
value: str = driver.status.questionable.get_enable()
```

Sets the bits of the ENABle part of the STATus:QUEStionable register. The enable part determines which events of the STATus:EVENt part are enabled for the summary bit in the status byte. These events can be used for a service request. If a bit in the ENABle part is 1, and the correesponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reportet to the next higher level.

**return**
enable: string

**get_event**() → str

```
# SCPI: STATus:QUEStionable:[EVENt]
value: str = driver.status.questionable.get_event()
```

Queries the content of the EVENt part of the method RsSmab.Status.Questionable.event register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**return**
value: No help available

**get_ntransition**() → str

```
# SCPI: STATus:QUEStionable:NTRansition
value: str = driver.status.questionable.get_ntransition()
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**return**
ntransition: string

**get_ptransition**() → str

```
# SCPI: STATus:QUEStionable:PTRansition
value: str = driver.status.questionable.get_ptransition()
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**return**
ptransition: string

**set_condition**(*condition: str*) → None

```
# SCPI: STATus:QUEStionable:CONDition
driver.status.questionable.set_condition(condition = 'abc')
```

Queries the content of the CONDition part of the STATus:QUEStionable register. This part contains information on the action currently being performed in the instrument. The content is not deleted after being read out since it indicates the current hardware status.

**param condition**
string

**set_enable**(*enable: str*) → None

```
# SCPI: STATus:QUEStionable:ENABle
driver.status.questionable.set_enable(enable = 'abc')
```

Sets the bits of the ENABle part of the STATus:QUEStionable register. The enable part determines which events of the STATus:EVENt part are enabled for the summary bit in the status byte. These events can be used for a service request. If a bit in the ENABle part is 1, and the correesponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reportet to the next higher level.

**param enable**
string

**set_event**(*value: str*) → None

```
# SCPI: STATus:QUEStionable:[EVENt]
driver.status.questionable.set_event(value = 'abc')
```

Queries the content of the EVENt part of the method RsSmab.Status.Questionable.event register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**param value**
string

**set_ntransition**(*ntransition: str*) → None

```
# SCPI: STATus:QUEStionable:NTRansition
driver.status.questionable.set_ntransition(ntransition = 'abc')
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**param ntransition**
string

**set_ptransition**(*ptransition: str*) → None

```
# SCPI: STATus:QUEStionable:PTRansition
driver.status.questionable.set_ptransition(ptransition = 'abc')
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**param ptransition**
string

#### Cloning the Group

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

#### Subgroups

##### 6.19.2.1 Bit<BitNumberNull>

#### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.status.questionable.bit.repcap_bitNumberNull_get()
driver.status.questionable.bit.repcap_bitNumberNull_set(repcap.BitNumberNull.Nr0)
```

**class BitCls**

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: BitNumberNull, default value after init: BitNumberNull.Nr0

#### Cloning the Group

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

#### Subgroups

##### 6.19.2.1.1 Condition

#### SCPI Command :

```
STATus:QUEStionable:BIT<BITNR>:CONDition
```

**class ConditionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:CONDition
value: str = driver.status.questionable.bit.condition.get(bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
condition: No help available

#### 6.19.2.1.2 Enable

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:ENABle
```

**class EnableCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:ENABle
value: str = driver.status.questionable.bit.enable.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
enable: No help available

**set**(*enable: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:ENABle
driver.status.questionable.bit.enable.set(enable = 'abc', bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param enable**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.19.2.1.3 Event

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:[EVENt]
```

**class EventCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:[EVENt]
value: str = driver.status.questionable.bit.event.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
event: No help available

#### 6.19.2.1.4 Ntransition

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:NTRansition
```

**class NtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:NTRansition
value: str = driver.status.questionable.bit.ntransition.get(bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ntransition: No help available

**set**(*ntransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:NTRansition
driver.status.questionable.bit.ntransition.set(ntransition = 'abc',␣
↪bitNumberNull = repcap.BitNumberNull.Default)
```

No command help available

**param ntransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.19.2.1.5 Ptransition

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:PTRansition
```

**class PtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:PTRansition
value: str = driver.status.questionable.bit.ptransition.get(bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ptransition: No help available

**set**(*ptransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:PTRansition
driver.status.questionable.bit.ptransition.set(ptransition = 'abc',␣
↪bitNumberNull = repcap.BitNumberNull.Default)
```

No command help available

**param ptransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

### 6.19.3 Queue

#### SCPI Command :

```
STATus:QUEue:[NEXT]
```

**class QueueCls**

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

**get_next**() → str

```
# SCPI: STATus:QUEue:[NEXT]
value: str = driver.status.queue.get_next()
```

Queries the oldest entry in the error queue and then deletes it. Positive error numbers denote device-specific errors, and negative error numbers denote error messages defined by SCPI. If the error queue is empty, 0 ('No error') is returned. The command is identical to SYSTem:ERRor[:NEXT]?.

**return**
next_py: string

## 6.20 Sweep

#### SCPI Command :

```
SWEep:TYPE
```

**class SweepCls**

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

**get_type_py**() → SweepType

```
# SCPI: SWEep:TYPE
value: enums.SweepType = driver.sweep.get_type_py()
```

Provided for compatibility between SCPI and Rohde & Schwarz commands.

**return**
sweep_type: ADVanced| STANdard

**set_type_py**(*sweep_type: SweepType*) → None

```
# SCPI: SWEep:TYPE
driver.sweep.set_type_py(sweep_type = enums.SweepType.ADVanced)
```

Provided for compatibility between SCPI and Rohde & Schwarz commands.

**param sweep_type**
ADVanced| STANdard

## 6.21 System

### SCPI Commands :

```
SYSTem:CRASh
SYSTem:DID
SYSTem:DLOCk
SYSTem:IMPort
SYSTem:IRESponse
SYSTem:KLOCk
SYSTem:LANGuage
SYSTem:NINFormation
SYSTem:ORESponse
SYSTem:OSYStem
SYSTem:PRESet
SYSTem:PRESet:ALL
SYSTem:PRESet:BASE
SYSTem:RCL
SYSTem:RESet
SYSTem:RESet:ALL
SYSTem:RESet:BASE
SYSTem:SAV
SYSTem:SIMulation
SYSTem:SRCat
SYSTem:SREStore
SYSTem:SRMode
SYSTem:SRSel
SYSTem:SSAVe
SYSTem:TZONe
SYSTem:UPTime
SYSTem:VERSion
SYSTem:WAIT
```

class SystemCls

System commands group definition. 191 total commands, 37 Subgroups, 28 group commands

get_did() → str

```
# SCPI: SYSTem:DID
value: str = driver.system.get_did()
```

No command help available

**return**
pseudo_string: No help available

get_dlock() → bool

```
# SCPI: SYSTem:DLOCk
value: bool = driver.system.get_dlock()
```

Disables the manual operation over the display, including the front panel keyboard of the instrument.

**return**
disp_lock_stat: 1| ON| 0| OFF

get_iresponse() → str

```
# SCPI: SYSTem:IRESponse
value: str = driver.system.get_iresponse()
```

Defines the user defined identification string for *IDN. Note: While working in an emulation mode, the instrument's specific command set is disabled, i.e. the SCPI command method RsSmab.System.iresponse is discarded.

**return**
idn_response: string

get_klock() → bool

```
# SCPI: SYSTem:KLOCk
value: bool = driver.system.get_klock()
```

Disables the front panel keyboard of the instrument.

**return**
state: 1| ON| 0| OFF

get_language() → str

```
# SCPI: SYSTem:LANGuage
value: str = driver.system.get_language()
```

Sets the remote control command set.

**return**
language: string

get_ninformation() → str

```
# SCPI: SYSTem:NINFormation
value: str = driver.system.get_ninformation()
```

Queries the oldest information message (‘Error History > Level > Info’) in the error/event queue.

**return**
next_info: string

**get_oresponse**() → str

```
# SCPI: SYSTem:ORESponse
value: str = driver.system.get_oresponse()
```

Defines the user defined response string for *OPT. Note: While working in an emulation mode, the instrument’s specific command set is disabled, i.e. the SCPI command method RsSmab.System.oresponse is discarded.

**return**
oresponse: string

**get_osystem**() → str

```
# SCPI: SYSTem:OSYStem
value: str = driver.system.get_osystem()
```

Queries the operating system of the instrument.

**return**
oper_system: string

**get_simulation**() → bool

```
# SCPI: SYSTem:SIMulation
value: bool = driver.system.get_simulation()
```

No command help available

**return**
status: No help available

**get_sr_cat**() → List[str]

```
# SCPI: SYSTem:SRCat
value: List[str] = driver.system.get_sr_cat()
```

No command help available

**return**
catalog: No help available

**get_sr_mode**() → RecScpiCmdMode

```
# SCPI: SYSTem:SRMode
value: enums.RecScpiCmdMode = driver.system.get_sr_mode()
```

No command help available

**return**
mode: No help available

**get_sr_sel**() → str

```
# SCPI: SYSTem:SRSel
value: str = driver.system.get_sr_sel()
```

No command help available

**return**
filename: No help available

**get_tzone**() → str

```
# SCPI: SYSTem:TZONe
value: str = driver.system.get_tzone()
```

No command help available

**return**
pseudo_string: No help available

**get_up_time**() → str

```
# SCPI: SYSTem:UPTime
value: str = driver.system.get_up_time()
```

Queries the up time of the operating system.

**return**
up_time: 'ddd.hh:mm:ss'

**get_version**() → str

```
# SCPI: SYSTem:VERSion
value: str = driver.system.get_version()
```

Queries the SCPI version the instrument's command set complies with.

**return**
version: string

**preset**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:PRESet
driver.system.preset(pseudo_string = 'abc')


    INTRO_CMD_HELP: Triggers an instrument reset. It has the same effect as:

    - The [Preset] key. However, the command does not close open GUI dialogs␣
↪like the key does.
    - The *RST command
```

For an overview of the settings affected by the preset function, see Table 'Key parameters affected by preset and factory preset'

**param pseudo_string**
No help available

**preset_all**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:PRESet:ALL
driver.system.preset_all(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**preset_base**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:PRESet:BASE
driver.system.preset_base(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**recall**(*path_name: str*) → None

```
# SCPI: SYSTem:RCL
driver.system.recall(path_name = 'abc')
```

Selects and uploads a *.savrcltxt file with previously saved R&S SMA100B settings from the default or a specified directory.

**param path_name**
string

**reset**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:RESet
driver.system.reset(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**reset_all**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:RESet:ALL
driver.system.reset_all(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**reset_base**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:RESet:BASE
driver.system.reset_base(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**save**(*path_name: str*) → None

```
# SCPI: SYSTem:SAV
driver.system.save(path_name = 'abc')
```

Saves the current R&S SMA100B settings in a file. To deteremine the file name and storage location, enter the directory and file name with the command. According to the file type, the R&S SMA100B assigns the extension (*. savrcltxt) automatically.

**param path_name**
string

**set_crash**(*test_scpi_generic: float*) → None

```
# SCPI: SYSTem:CRASh
driver.system.set_crash(test_scpi_generic = 1.0)
```

No command help available

**param test_scpi_generic**
No help available

**set_dlock**(*disp_lock_stat: bool*) → None

```
# SCPI: SYSTem:DLOCk
driver.system.set_dlock(disp_lock_stat = False)
```

Disables the manual operation over the display, including the front panel keyboard of the instrument.

**param disp_lock_stat**
1| ON| 0| OFF

**set_import_py**(*filename: str*) → None

```
# SCPI: SYSTem:IMPort
driver.system.set_import_py(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_iresponse**(*idn_response: str*) → None

```
# SCPI: SYSTem:IRESponse
driver.system.set_iresponse(idn_response = 'abc')
```

Defines the user defined identification string for *IDN. Note: While working in an emulation mode, the instrument's specific command set is disabled, i.e. the SCPI command method RsSmab.System.iresponse is discarded.

**param idn_response**
string

**set_klock**(*state: bool*) → None

```
# SCPI: SYSTem:KLOCk
driver.system.set_klock(state = False)
```

Disables the front panel keyboard of the instrument.

**param state**
1| ON| 0| OFF

**set_language**(*language: str*) → None

```
# SCPI: SYSTem:LANGuage
driver.system.set_language(language = 'abc')
```

Sets the remote control command set.

**param language**
string

**set_oresponse**(*oresponse: str*) → None

```
# SCPI: SYSTem:ORESponse
driver.system.set_oresponse(oresponse = 'abc')
```

Defines the user defined response string for *OPT. Note: While working in an emulation mode, the instrument's specific command set is disabled, i.e. the SCPI command method RsSmab.System.oresponse is discarded.

**param oresponse**
string

**set_sr_mode**(*mode: RecScpiCmdMode*) → None

```
# SCPI: SYSTem:SRMode
driver.system.set_sr_mode(mode = enums.RecScpiCmdMode.AUTO)
```

No command help available

**param mode**
No help available

**set_sr_sel**(*filename: str*) → None

```
# SCPI: SYSTem:SRSel
driver.system.set_sr_sel(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_srestore**(*data_set: int*) → None

```
# SCPI: SYSTem:SREStore
driver.system.set_srestore(data_set = 1)
```

No command help available

**param data_set**
No help available

**set_ssave**(*data_set: int*) → None

```
# SCPI: SYSTem:SSAVe
driver.system.set_ssave(data_set = 1)
```

No command help available

**param data_set**
No help available

**set_tzone**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TZONe
driver.system.set_tzone(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**set_wait**(*time_ms: int*) → None

```
# SCPI: SYSTem:WAIT
driver.system.set_wait(time_ms = 1)
```

Delays the execution of the subsequent remote command by the specified time. This function is useful, for example to execute an SCPI sequence automatically but with a defined time delay between some commands. See 'How to assign actions to the [ (User) ] key'.

**param time_ms**
integer Wait time in ms Range: 0 to 10000

#### Cloning the Group

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

#### Subgroups

### 6.21.1 Beeper

#### SCPI Command :

```
SYSTem:BEEPer:STATe
```

**class BeeperCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:BEEPer:STATe
value: bool = driver.system.beeper.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:BEEPer:STATe
driver.system.beeper.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.21.2 Bios

#### SCPI Command :

```
SYSTem:BIOS:VERSion
```

**class BiosCls**

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

**get_version**() → str

```
# SCPI: SYSTem:BIOS:VERSion
value: str = driver.system.bios.get_version()
```

Queries the BIOS version of the instrument.

**return**
version: string

### 6.21.3 Communicate

**class CommunicateCls**

Communicate commands group definition. 23 total commands, 7 Subgroups, 0 group commands

#### Cloning the Group

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

### Subgroups

#### 6.21.3.1 Gpib

### SCPI Commands :

```
SYSTem:COMMunicate:GPIB:LTERminator
SYSTem:COMMunicate:GPIB:RESource
```

**class GpibCls**

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

**get_lterminator**() → IecTermMode

```
# SCPI: SYSTem:COMMunicate:GPIB:LTERminator
value: enums.IecTermMode = driver.system.communicate.gpib.get_lterminator()
```

Sets the terminator recognition for remote control via GPIB interface.

**return**
lterminator: STANdard| EOI EOI Recognizes an LF (Line Feed) as the terminator only when it is sent with the line message EOI (End of Line) . This setting is recommended particularly for binary block transmissions, as binary blocks may coincidentally contain a characater with value LF (Line Feed) , although it is not determined as a terminator. STANdard Recognizes an LF (Line Feed) as the terminator regardless of whether it is sent with or without EOI.

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:GPIB:RESource
value: str = driver.system.communicate.gpib.get_resource()
```

Queries the visa resource string for remote control via the GPIB interface. To change the GPIB address, use the command method RsSmab.System.Communicate.Gpib.Self.address.

**return**
resource: string

**set_lterminator**(*lterminator: IecTermMode*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB:LTERminator
driver.system.communicate.gpib.set_lterminator(lterminator = enums.IecTermMode.
↪EOI)
```

Sets the terminator recognition for remote control via GPIB interface.

**param lterminator**
STANdard| EOI EOI Recognizes an LF (Line Feed) as the terminator only when it is sent with the line message EOI (End of Line) . This setting is recommended particularly for binary block transmissions, as binary blocks may coincidentally contain a characater with value LF (Line Feed) , although it is not determined as a terminator. STANdard Recognizes an LF (Line Feed) as the terminator regardless of whether it is sent with or without EOI.

#### Cloning the Group

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

#### Subgroups

##### 6.21.3.1.1 Self

#### SCPI Command :

```
SYSTem:COMMunicate:GPIB:[SELF]:ADDRess
```

**class SelfCls**

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

**get_address**() → int

```
# SCPI: SYSTem:COMMunicate:GPIB:[SELF]:ADDRess
value: int = driver.system.communicate.gpib.self.get_address()
```

Sets the GPIB address.

**return**
address: integer Range: 0 to 30

**set_address**(*address: int*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB:[SELF]:ADDRess
driver.system.communicate.gpib.self.set_address(address = 1)
```

Sets the GPIB address.

**param address**
integer Range: 0 to 30

##### 6.21.3.2 Hislip

#### SCPI Command :

```
SYSTem:COMMunicate:HISLip:RESource
```

**class HislipCls**

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

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:HISLip:RESource
value: str = driver.system.communicate.hislip.get_resource()
```

Queries the VISA resource string. This string is used for remote control of the instrument with HiSLIP protocol.

**return**
resource: string

#### 6.21.3.3 Network

**SCPI Commands :**

```
SYSTem:COMMunicate:NETWork:MACaddress
SYSTem:COMMunicate:NETWork:RESource
SYSTem:COMMunicate:NETWork:STATus
```

**class NetworkCls**

Network commands group definition. 12 total commands, 3 Subgroups, 3 group commands

**get_mac_address**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:MACaddress
value: str = driver.system.communicate.network.get_mac_address()
```

Queries the MAC address of the network adapter. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State.set.

**return**
mac_address: string

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:RESource
value: str = driver.system.communicate.network.get_resource()
```

Queries the visa resource string for Ethernet instruments.

**return**
resource: string

**get_status**() → bool

```
# SCPI: SYSTem:COMMunicate:NETWork:STATus
value: bool = driver.system.communicate.network.get_status()
```

Queries the network configuration state.

**return**
state: 1| ON| 0| OFF

**set_mac_address**(*mac_address: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:MACaddress
driver.system.communicate.network.set_mac_address(mac_address = 'abc')
```

Queries the MAC address of the network adapter. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State.set.

**param mac_address**
string

#### Cloning the Group

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

#### Subgroups

##### 6.21.3.3.1 Common

#### SCPI Commands :

```
SYSTem:COMMunicate:NETWork:[COMMon]:DOMain
SYSTem:COMMunicate:NETWork:[COMMon]:HOSTname
SYSTem:COMMunicate:NETWork:[COMMon]:WORKgroup
```

**class CommonCls**

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

**get_domain**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:DOMain
value: str = driver.system.communicate.network.common.get_domain()
```

Determines the primary suffix of the network domain.

**return**
domain: string

**get_hostname**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:HOSTname
value: str = driver.system.communicate.network.common.get_hostname()
```

Sets an individual hostname for the Signal Generator. Note:We recommend that you do not change the hostname to avoid problems with the network connection. If you change the hostname, be sure to use a unique name. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State. set.

**return**
hostname: string

**get_workgroup**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:WORKgroup
value: str = driver.system.communicate.network.common.get_workgroup()
```

Sets an individual workgroup name for the instrument.

**return**
workgroup: string

**set_domain**(*domain: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:DOMain
driver.system.communicate.network.common.set_domain(domain = 'abc')
```

Determines the primary suffix of the network domain.

**param domain**
string

**set_hostname**(*hostname: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:HOSTname
driver.system.communicate.network.common.set_hostname(hostname = 'abc')
```

Sets an individual hostname for the Signal Generator. Note:We recommend that you do not change the hostname to avoid problems with the network connection. If you change the hostname, be sure to use a unique name. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State. set.

**param hostname**
string

**set_workgroup**(*workgroup: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:WORKgroup
driver.system.communicate.network.common.set_workgroup(workgroup = 'abc')
```

Sets an individual workgroup name for the instrument.

**param workgroup**
string

#### 6.21.3.3.2 IpAddress

##### SCPI Commands :

```
SYSTem:COMMunicate:NETWork:IPADdress:MODE
SYSTem:COMMunicate:NETWork:[IPADdress]:DNS
SYSTem:COMMunicate:NETWork:[IPADdress]:GATeway
SYSTem:COMMunicate:NETWork:IPADdress
```

**class IpAddressCls**

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

**get_dns**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:DNS
value: str = driver.system.communicate.network.ipAddress.get_dns()
```

Determines or queries the network DNS server to resolve the name.

**return**
dns: string

**get_gateway**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:GATeway
value: str = driver.system.communicate.network.ipAddress.get_gateway()
```

Sets the IP address of the default gateway.

**return**
gateway: string Range: 0.0.0.0 to ff.ff.ff.ff

**get_mode**() → NetMode

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress:MODE
value: enums.NetMode = driver.system.communicate.network.ipAddress.get_mode()
```

Selects manual or automatic setting of the IP address.

**return**
mode: AUTO| STATic

**get_value**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress
value: str = driver.system.communicate.network.ipAddress.get_value()
```

Sets the IP address.

**return**
ip_address: string Range: 0.0.0.0. to ff.ff.ff.ff

**set_dns**(*dns: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:DNS
driver.system.communicate.network.ipAddress.set_dns(dns = 'abc')
```

Determines or queries the network DNS server to resolve the name.

**param dns**
string

**set_gateway**(*gateway: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:GATeway
driver.system.communicate.network.ipAddress.set_gateway(gateway = 'abc')
```

Sets the IP address of the default gateway.

**param gateway**
string Range: 0.0.0.0 to ff.ff.ff.ff

**set_mode**(*mode: NetMode*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress:MODE
driver.system.communicate.network.ipAddress.set_mode(mode = enums.NetMode.AUTO)
```

Selects manual or automatic setting of the IP address.

**param mode**
AUTO| STATic

**set_value**(*ip_address: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress
driver.system.communicate.network.ipAddress.set_value(ip_address = 'abc')
```

Sets the IP address.

**param ip_address**
string Range: 0.0.0.0. to ff.ff.ff.ff

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.communicate.network.ipAddress.clone()
```

### Subgroups

#### 6.21.3.3.2.1 Subnet

### SCPI Command :

```
SYSTem:COMMunicate:NETWork:[IPADdress]:SUBNet:MASK
```

**class SubnetCls**

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

**get_mask**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:SUBNet:MASK
value: str = driver.system.communicate.network.ipAddress.subnet.get_mask()
```

Sets the subnet mask.

**return**
mask: string

**set_mask**(*mask: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:SUBNet:MASK
driver.system.communicate.network.ipAddress.subnet.set_mask(mask = 'abc')
```

Sets the subnet mask.

**param mask**
string

#### 6.21.3.3.3 Restart

### SCPI Command :

```
SYSTem:COMMunicate:NETWork:RESTart
```

**class RestartCls**

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

**set**() → None

```
# SCPI: SYSTem:COMMunicate:NETWork:RESTart
driver.system.communicate.network.restart.set()
```

Restarts the network.

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

```
# SCPI: SYSTem:COMMunicate:NETWork:RESTart
driver.system.communicate.network.restart.set_with_opc()
```

Restarts the network.

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

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

#### 6.21.3.4 Scpi

**class ScpiCls**

Scpi 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.system.communicate.scpi.clone()
```

##### Subgroups

###### 6.21.3.4.1 Ethernet

##### SCPI Command :

```
SYSTem:COMMunicate:SCPI:ETHernet:[ACTive]
```

**class EthernetCls**

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

**get_active**() → str

```
# SCPI: SYSTem:COMMunicate:SCPI:ETHernet:[ACTive]
value: str = driver.system.communicate.scpi.ethernet.get_active()
```

No command help available

**return**
active_connection: No help available

#### 6.21.3.5 Serial

**SCPI Commands :**

```
SYSTem:COMMunicate:SERial:BAUD
SYSTem:COMMunicate:SERial:PARity
SYSTem:COMMunicate:SERial:RESource
SYSTem:COMMunicate:SERial:SBITs
```

**class SerialCls**

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

**get_baud**() → Rs232BdRate

```
# SCPI: SYSTem:COMMunicate:SERial:BAUD
value: enums.Rs232BdRate = driver.system.communicate.serial.get_baud()
```

Defines the baudrate for the serial remote control interface.

**return**
baud: 2400| 4800| 9600| 19200| 38400| 57600| 115200

**get_parity**() → Parity

```
# SCPI: SYSTem:COMMunicate:SERial:PARity
value: enums.Parity = driver.system.communicate.serial.get_parity()
```

Enters the parity for the serial remote control interface.

**return**
parity: NONE| ODD| EVEN

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:SERial:RESource
value: str = driver.system.communicate.serial.get_resource()
```

Queries the visa resource string for the serial remote control interface. This string is used for remote control of the instrument.

**return**
resource: string

**get_sbits**() → Rs232StopBits

```
# SCPI: SYSTem:COMMunicate:SERial:SBITs
value: enums.Rs232StopBits = driver.system.communicate.serial.get_sbits()
```

Defines the number of stop bits for the serial remote control interface.

**return**
sbits: 1| 2

**set_baud**(*baud: Rs232BdRate*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:BAUD
driver.system.communicate.serial.set_baud(baud = enums.Rs232BdRate._115200)
```

Defines the baudrate for the serial remote control interface.

**param baud**
2400| 4800| 9600| 19200| 38400| 57600| 115200

**set_parity**(*parity: Parity*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:PARity
driver.system.communicate.serial.set_parity(parity = enums.Parity.EVEN)
```

Enters the parity for the serial remote control interface.

**param parity**
NONE| ODD| EVEN

**set_sbits**(*sbits: Rs232StopBits*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:SBITs
driver.system.communicate.serial.set_sbits(sbits = enums.Rs232StopBits._1)
```

Defines the number of stop bits for the serial remote control interface.

**param sbits**
1| 2

#### 6.21.3.6 Socket

**SCPI Command :**

```
SYSTem:COMMunicate:SOCKet:RESource
```

**class SocketCls**

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

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:SOCKet:RESource
value: str = driver.system.communicate.socket.get_resource()
```

Queries the visa resource string for remote control via LAN interface, using TCP/IP socket protocol.

**return**
resource: string

#### 6.21.3.7 Usb

**SCPI Command :**

```
SYSTem:COMMunicate:USB:RESource
```

**class UsbCls**

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

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:USB:RESource
value: str = driver.system.communicate.usb.get_resource()
```

Queries the visa resource string for remote control via the USB interface.

**return**
resource: string

### 6.21.4 Date

#### SCPI Commands :

```
SYSTem:DATE
SYSTem:DATE:LOCal
SYSTem:DATE:UTC
```

**class DateCls**

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

**class DateStruct**

Response structure. Fields:

- Year: List[int]: integer
- Month: int: integer Range: 1 to 12
- Day: int: integer Range: 1 to 31

**get**() → DateStruct

```
# SCPI: SYSTem:DATE
value: DateStruct = driver.system.date.get()
```

Queries or sets the date for the instrument-internal calendar. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State.set.

**return**
structure: for return value, see the help for DateStruct structure arguments.

**get_local**() → str

```
# SCPI: SYSTem:DATE:LOCal
value: str = driver.system.date.get_local()
```

No command help available

**return**
pseudo_string: No help available

**get_utc**() → str

```
# SCPI: SYSTem:DATE:UTC
value: str = driver.system.date.get_utc()
```

No command help available

**return**
pseudo_string: No help available

**set**(*year: List[int]*, *month: int*, *day: int*) → None

```
# SCPI: SYSTem:DATE
driver.system.date.set(year = [1, 2, 3], month = 1, day = 1)
```

Queries or sets the date for the instrument-internal calendar. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State.set.

**param year**
integer

**param month**
integer Range: 1 to 12

**param day**
integer Range: 1 to 31

**set_local**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:DATE:LOCal
driver.system.date.set_local(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**set_utc**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:DATE:UTC
driver.system.date.set_utc(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

### 6.21.5 Device

#### SCPI Command :

```
SYSTem:DEVice:ID
```

**class DeviceCls**

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

**get_id**() → str

```
# SCPI: SYSTem:DEVice:ID
value: str = driver.system.device.get_id()
```

No command help available

**return**
pseudo_string: No help available

### 6.21.6 DeviceFootprint

#### SCPI Command :

```
SYSTem:DFPRint
```

**class DeviceFootprintCls**

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

**get**() → str

```
# SCPI: SYSTem:DFPRint
value: str = driver.system.deviceFootprint.get()
```

Queries the device footprint of the instrument. The retrieved information is in machine-readable form suitable for automatic further processing.

**return**
device_footprint: string Information on the instrument type, device identification and details on the installed FW version, hardware and software options.

**set**(*directory: str*) → None

```
# SCPI: SYSTem:DFPRint
driver.system.deviceFootprint.set(directory = 'abc')
```

Queries the device footprint of the instrument. The retrieved information is in machine-readable form suitable for automatic further processing.

**param directory**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.21.6.1 History

#### SCPI Commands :

```
SYSTem:DFPRint:HISTory:COUNt
SYSTem:DFPRint:HISTory:ENTRy
```

**class HistoryCls**

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

**get_count**() → str

```
# SCPI: SYSTem:DFPRint:HISTory:COUNt
value: str = driver.system.deviceFootprint.history.get_count()
```

No command help available

**return**
pseudo_string: No help available

**get_entry**() → str

```
# SCPI: SYSTem:DFPRint:HISTory:ENTRy
value: str = driver.system.deviceFootprint.history.get_entry()
```

No command help available

**return**
pseudo_string: No help available

### 6.21.7 Dexchange

#### SCPI Commands :

```
SYSTem:DEXChange:CATalog
SYSTem:DEXChange:DEBug
SYSTem:DEXChange:DELete
SYSTem:DEXChange:FORMat
SYSTem:DEXChange:SELect
```

**class DexchangeCls**

Dexchange commands group definition. 12 total commands, 3 Subgroups, 5 group commands

**delete**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:DELete
driver.system.dexchange.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**get_catalog**() → List[str]

```
# SCPI: SYSTem:DEXChange:CATalog
value: List[str] = driver.system.dexchange.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_debug**() → bool

```
# SCPI: SYSTem:DEXChange:DEBug
value: bool = driver.system.dexchange.get_debug()
```

No command help available

**return**
debug: No help available

**get_format_py**() → DevExpFormat

```
# SCPI: SYSTem:DEXChange:FORMat
value: enums.DevExpFormat = driver.system.dexchange.get_format_py()
```

No command help available

**return**
format_py: No help available

**get_select**() → str

```
# SCPI: SYSTem:DEXChange:SELect
value: str = driver.system.dexchange.get_select()
```

No command help available

**return**
filename: No help available

**set_debug**(*debug: bool*) → None

```
# SCPI: SYSTem:DEXChange:DEBug
driver.system.dexchange.set_debug(debug = False)
```

No command help available

**param debug**
No help available

**set_format_py**(*format_py: DevExpFormat*) → None

```
# SCPI: SYSTem:DEXChange:FORMat
driver.system.dexchange.set_format_py(format_py = enums.DevExpFormat.
↪CGPRedefined)
```

No command help available

**param format_py**
No help available

**set_select**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:SELect
driver.system.dexchange.set_select(filename = 'abc')
```

No command help available

**param filename**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.7.1 Execute

### SCPI Command :

```
SYSTem:DEXChange:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: SYSTem:DEXChange:EXECute
driver.system.dexchange.execute.set()
```

No command help available

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

```
# SCPI: SYSTem:DEXChange:EXECute
driver.system.dexchange.execute.set_with_opc()
```

No command help available

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

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

#### 6.21.7.2 Template

**class TemplateCls**

Template commands group definition. 5 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.21.7.2.1 Predefined

### SCPI Commands :

```
SYSTem:DEXChange:TEMPlate:PREDefined:CATalog
SYSTem:DEXChange:TEMPlate:PREDefined:SELect
```

**class PredefinedCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:DEXChange:TEMPlate:PREDefined:CATalog
value: List[str] = driver.system.dexchange.template.predefined.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_select**() → str

```
# SCPI: SYSTem:DEXChange:TEMPlate:PREDefined:SELect
value: str = driver.system.dexchange.template.predefined.get_select()
```

No command help available

**return**
filename: No help available

**set_select**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:TEMPlate:PREDefined:SELect
driver.system.dexchange.template.predefined.set_select(filename = 'abc')
```

No command help available

**param filename**
No help available

#### 6.21.7.2.2 User

### SCPI Commands :

```
SYSTem:DEXChange:TEMPlate:USER:CATalog
SYSTem:DEXChange:TEMPlate:USER:DELete
SYSTem:DEXChange:TEMPlate:USER:SELect
```

**class UserCls**

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

**delete**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:DELete
driver.system.dexchange.template.user.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**get_catalog**() → List[str]

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:CATalog
value: List[str] = driver.system.dexchange.template.user.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_select**() → str

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:SELect
value: str = driver.system.dexchange.template.user.get_select()
```

No command help available

**return**
filename: No help available

**set_select**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:SELect
driver.system.dexchange.template.user.set_select(filename = 'abc')
```

No command help available

**param filename**
No help available

#### 6.21.7.3 Transaction

**SCPI Command :**

```
SYSTem:DEXChange:TRANsaction:STATe
```

**class TransactionCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:DEXChange:TRANsaction:STATe
value: bool = driver.system.dexchange.transaction.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:DEXChange:TRANsaction:STATe
driver.system.dexchange.transaction.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.21.8 Error

#### SCPI Commands :

```
SYSTem:ERRor:ALL
SYSTem:ERRor:COUNt
SYSTem:ERRor:STATic
```

**class ErrorCls**

Error commands group definition. 7 total commands, 2 Subgroups, 3 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.

**return**
all_py: string Error/event_number,'Error/event_description[;Device-dependent info]' A comma separated list of error number and a short description of the error in FIFO order. If the queue is empty, the response is 0,'No error' Positive error numbers are instrument-dependent. Negative error numbers are reserved by the SCPI standard. Volatile errors are reported once, at the time they appear. Identical errors are reported repeatedly only if the original error has already been retrieved from (and hence not any more present in) the error queue.

**get_count**() → str

```
# SCPI: SYSTem:ERRor:COUNt
value: str = driver.system.error.get_count()
```

Queries the number of entries in the error queue.

**return**
count: integer 0 The error queue is empty.

**get_static**() → str

```
# SCPI: SYSTem:ERRor:STATic
value: str = driver.system.error.get_static()
```

Returns a list of all errors existing at the time when the query is started. This list corresponds to the display on the info page under manual control.

**return**
static_errors: string

### Cloning the Group

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

### Subgroups

#### 6.21.8.1 Code

### SCPI Commands :

```
SYSTem:ERRor:CODE:ALL
SYSTem:ERRor:CODE:[NEXT]
```

**class CodeCls**

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

**get_all**() → str

```
# SCPI: SYSTem:ERRor:CODE:ALL
value: str = driver.system.error.code.get_all()
```

Queries the error numbers of all entries in the error queue and then deletes them.

**return**
all_py: string Returns the error numbers. To retrieve the entire error text, send the command method RsSmab.System.Error.all. 0 'No error', i.e. the error queue is empty Positive value Positive error numbers denote device-specific errors Negative value Negative error numbers denote error messages defined by SCPI.

**get_next**() → str

```
# SCPI: SYSTem:ERRor:CODE:[NEXT]
value: str = driver.system.error.code.get_next()
```

Queries the error number of the oldest entry in the error queue and then deletes it.

**return**
next_py: string Returns the error number. To retrieve the entire error text, send the command method RsSmab.System.Error.all. 0 'No error', i.e. the error queue is empty Positive value Positive error numbers denote device-specific errors Negative value Negative error numbers denote error messages defined by SCPI.

#### 6.21.8.2 History

**SCPI Commands :**

```
SYSTem:ERRor:HISTory:CLEar
SYSTem:ERRor:HISTory
```

**class HistoryCls**

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

**clear**() → None

```
# SCPI: SYSTem:ERRor:HISTory:CLEar
driver.system.error.history.clear()
```

Clears the error history.

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

```
# SCPI: SYSTem:ERRor:HISTory:CLEar
driver.system.error.history.clear_with_opc()
```

Clears the error history.

Same as clear, but waits for the operation to complete before continuing further. Use the RsSmab.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: SYSTem:ERRor:HISTory
value: str = driver.system.error.history.get_value()
```

No command help available

**return**
error_history: No help available

### 6.21.9 ExtDevices

**class ExtDevicesCls**

ExtDevices commands group definition. 5 total commands, 1 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.21.9.1 Update

#### SCPI Command :

```
SYSTem:EXTDevices:UPDate
```

**class UpdateCls**

Update commands group definition. 5 total commands, 3 Subgroups, 1 group commands

**set**() → None

```
# SCPI: SYSTem:EXTDevices:UPDate
driver.system.extDevices.update.set()
```

No command help available

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

```
# SCPI: SYSTem:EXTDevices:UPDate
driver.system.extDevices.update.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsSmab.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.system.extDevices.update.clone()
```

#### Subgroups

##### 6.21.9.1.1 Check

#### SCPI Command :

```
SYSTem:EXTDevices:UPDate:CHECk
```

**class CheckCls**

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

**set**() → None

```
# SCPI: SYSTem:EXTDevices:UPDate:CHECk
driver.system.extDevices.update.check.set()
```

No command help available

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

```
# SCPI: SYSTem:EXTDevices:UPDate:CHECk
driver.system.extDevices.update.check.set_with_opc()
```

No command help available

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

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

#### 6.21.9.1.2 Needed

**SCPI Command :**

```
SYSTem:EXTDevices:UPDate:NEEDed:[STATe]
```

**class NeededCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:EXTDevices:UPDate:NEEDed:[STATe]
value: bool = driver.system.extDevices.update.needed.get_state()
```

No command help available

**return**
update_needed: No help available

#### 6.21.9.1.3 Tselected

**SCPI Commands :**

```
SYSTem:EXTDevices:UPDate:TSELected:CATalog
SYSTem:EXTDevices:UPDate:TSELected:STEP
```

**class TselectedCls**

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

**get_catalog**() → str

```
# SCPI: SYSTem:EXTDevices:UPDate:TSELected:CATalog
value: str = driver.system.extDevices.update.tselected.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_step**() → str

```
# SCPI: SYSTem:EXTDevices:UPDate:TSELected:STEP
value: str = driver.system.extDevices.update.tselected.get_step()
```

No command help available

**return**
sel_string: No help available

**set_step**(*sel_string: str*) → None

```
# SCPI: SYSTem:EXTDevices:UPDate:TSELected:STEP
driver.system.extDevices.update.tselected.set_step(sel_string = 'abc')
```

No command help available

**param sel_string**
No help available

### 6.21.10 Files

**class FilesCls**

Files 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.system.files.clone()
```

#### Subgroups

##### 6.21.10.1 Temporary

#### SCPI Command :

```
SYSTem:FILes:TEMPorary:DELete
```

**class TemporaryCls**

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

**delete**() → None

```
# SCPI: SYSTem:FILes:TEMPorary:DELete
driver.system.files.temporary.delete()
```

Deletes the temporary files from the internal memory or, if installed, from the SD card slot.

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

```
# SCPI: SYSTem:FILes:TEMPorary:DELete
driver.system.files.temporary.delete_with_opc()
```

Deletes the temporary files from the internal memory or, if installed, from the SD card slot.

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

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

### 6.21.11 FpFpga

#### SCPI Command :

```
SYSTem:FPFPga:UPDate
```

**class FpFpgaCls**

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

**set_update**(*display_size: str*) → None

```
# SCPI: SYSTem:FPFPga:UPDate
driver.system.fpFpga.set_update(display_size = 'abc')
```

No command help available

**param display_size**
No help available

### 6.21.12 Fpreset

#### SCPI Command :

```
SYSTem:FPReset
```

**class FpresetCls**

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

**set**() → None

```
# SCPI: SYSTem:FPReset
driver.system.fpreset.set()
```

Triggers an instrument reset to the original state of delivery.

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

```
# SCPI: SYSTem:FPReset
driver.system.fpreset.set_with_opc()
```

Triggers an instrument reset to the original state of delivery.

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

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

### 6.21.13 Generic

#### SCPI Command :

```
SYSTem:GENeric:MSG
```

**class GenericCls**

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

**get_msg**() → str

```
# SCPI: SYSTem:GENeric:MSG
value: str = driver.system.generic.get_msg()
```

No command help available

**return**
generic_message: No help available

**set_msg**(*generic_message: str*) → None

```
# SCPI: SYSTem:GENeric:MSG
driver.system.generic.set_msg(generic_message = 'abc')
```

No command help available

**param generic_message**
No help available

### 6.21.14 Help

#### SCPI Commands :

```
SYSTem:HELP:EXPort
SYSTem:HELP:HEADers
```

**class HelpCls**

Help commands group definition. 4 total commands, 1 Subgroups, 2 group commands

**export**() → None

```
# SCPI: SYSTem:HELP:EXPort
driver.system.help.export()
```

Saves the online help as zip archive in the user directory.

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

```
# SCPI: SYSTem:HELP:EXPort
driver.system.help.export_with_opc()
```

Saves the online help as zip archive in the user directory.

Same as export, but waits for the operation to complete before continuing further. Use the RsSmab.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_headers**() → str

```
# SCPI: SYSTem:HELP:HEADers
value: str = driver.system.help.get_headers()
```

No command help available

**return**
headers: No help available

### Cloning the Group

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

### Subgroups

#### 6.21.14.1 Syntax

### SCPI Commands :

```
SYSTem:HELP:SYNTax:ALL
SYSTem:HELP:SYNTax
```

**class SyntaxCls**

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

**get_all**() → str

```
# SCPI: SYSTem:HELP:SYNTax:ALL
value: str = driver.system.help.syntax.get_all()
```

No command help available

**return**
pseudo_string: No help available

**get_value**() → str

```
# SCPI: SYSTem:HELP:SYNTax
value: str = driver.system.help.syntax.get_value()
```

No command help available

**return**
pseudo_string: No help available

### 6.21.15 Identification

#### SCPI Commands :

```
SYSTem:IDENtification:PRESet
SYSTem:IDENtification
```

**class IdentificationCls**

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

**get_value**() → IecDevId

```
# SCPI: SYSTem:IDENtification
value: enums.IecDevId = driver.system.identification.get_value()
```

Selects the mode to determine the 'IDN String' and the 'OPT String' for the instrument, selected with command method RsSmab.System.language. Note: While working in an emulation mode, the R&S SMA100B specific command set is disabled, that is, the SCPI command method RsSmab.System.Identification.value is discarded.

**return**
identification: AUTO| USER AUTO Automatically determines the strings. USER User-defined strings can be selected.

**preset**() → None

```
# SCPI: SYSTem:IDENtification:PRESet
driver.system.identification.preset()
```

Sets the *IDN and *OPT strings in user defined mode to default values.

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

```
# SCPI: SYSTem:IDENtification:PRESet
driver.system.identification.preset_with_opc()
```

Sets the *IDN and *OPT strings in user defined mode to default values.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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**(*identification: IecDevId*) → None

```
# SCPI: SYSTem:IDENtification
driver.system.identification.set_value(identification = enums.IecDevId.AUTO)
```

Selects the mode to determine the 'IDN String' and the 'OPT String' for the instrument, selected with command method RsSmab.System.language. Note: While working in an emulation mode, the R&S SMA100B specific command set is disabled, that is, the SCPI command method RsSmab.System.Identification.value is discarded.

**param identification**
AUTO| USER AUTO Automatically determines the strings. USER User-defined strings can be selected.

### 6.21.16 Information

#### SCPI Command :

```
SYSTem:INFormation:SR
```

**class InformationCls**

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

**get_sr**() → str

```
# SCPI: SYSTem:INFormation:SR
value: str = driver.system.information.get_sr()
```

No command help available

**return**
sr_info: No help available

**set_sr**(*sr_info: str*) → None

```
# SCPI: SYSTem:INFormation:SR
driver.system.information.set_sr(sr_info = 'abc')
```

No command help available

**param sr_info**
No help available

### 6.21.17 Linux

**class LinuxCls**

Linux 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.system.linux.clone()
```

#### Subgroups

##### 6.21.17.1 Kernel

#### SCPI Command :

```
SYSTem:LINux:KERNel:VERSion
```

**class KernelCls**

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

**get_version**() → str

```
# SCPI: SYSTem:LINux:KERNel:VERSion
value: str = driver.system.linux.kernel.get_version()
```

No command help available

**return**
version: No help available

### 6.21.18 Lock

#### SCPI Command :

```
SYSTem:LOCK:TIMeout
```

**class LockCls**

Lock commands group definition. 10 total commands, 5 Subgroups, 1 group commands

**get_timeout**() → int

```
# SCPI: SYSTem:LOCK:TIMeout
value: int = driver.system.lock.get_timeout()
```

No command help available

**return**
time_ms: No help available

**set_timeout**(*time_ms: int*) → None

```
# SCPI: SYSTem:LOCK:TIMeout
driver.system.lock.set_timeout(time_ms = 1)
```

No command help available

**param time_ms**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.18.1 Name

### SCPI Commands :

```
SYSTem:LOCK:NAME:DETailed
SYSTem:LOCK:NAME
```

**class NameCls**

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

**get_detailed**() → str

```
# SCPI: SYSTem:LOCK:NAME:DETailed
value: str = driver.system.lock.name.get_detailed()
```

No command help available

**return**
details: No help available

**get_value**() → str

```
# SCPI: SYSTem:LOCK:NAME
value: str = driver.system.lock.name.get_value()
```

No command help available

**return**
name: No help available

#### 6.21.18.2 Owner

### SCPI Commands :

```
SYSTem:LOCK:OWNer:DETailed
SYSTem:LOCK:OWNer
```

**class OwnerCls**

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

**get_detailed**() → str

```
# SCPI: SYSTem:LOCK:OWNer:DETailed
value: str = driver.system.lock.owner.get_detailed()
```

No command help available

**return**
details: No help available

**get_value**() → str

```
# SCPI: SYSTem:LOCK:OWNer
value: str = driver.system.lock.owner.get_value()
```

Queries the sessions that have locked the instrument currently. If an exclusive lock is set, the query returns the owner of this exclusive lock, otherwise it returns NONE.

**return**
owner: string

#### 6.21.18.3 Release

**SCPI Commands :**

```
SYSTem:LOCK:RELease:ALL
SYSTem:LOCK:RELease
```

**class ReleaseCls**

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

**set_all**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:LOCK:RELease:ALL
driver.system.lock.release.set_all(pseudo_string = 'abc')
```

Revokes the exclusive access to the instrument.

**param pseudo_string**
No help available

**set_value**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:LOCK:RELease
driver.system.lock.release.set_value(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

#### 6.21.18.4 Request

**SCPI Command :**

```
SYSTem:LOCK:REQuest:[EXCLusive]
```

**class RequestCls**

Request commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_exclusive**() → int

```
# SCPI: SYSTem:LOCK:REQuest:[EXCLusive]
value: int = driver.system.lock.request.get_exclusive()
```

Queries whether a lock for exclusive access to the instrument via ethernet exists. If successful, the query returns a 1, otherwise 0.

**return**
success: integer

### Cloning the Group

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

### Subgroups

#### 6.21.18.4.1 Shared

### SCPI Command :

```
SYSTem:LOCK:REQuest:SHARed
```

**class SharedCls**

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

**get**(*name: str*, *timeout_ms: int*) → int

```
# SCPI: SYSTem:LOCK:REQuest:SHARed
value: int = driver.system.lock.request.shared.get(name = 'abc', timeout_ms = 1)
```

No command help available

**param name**
No help available

**param timeout_ms**
No help available

**return**
success: No help available

#### 6.21.18.5 Shared

### SCPI Command :

```
SYSTem:LOCK:SHARed:STRing
```

**class SharedCls**

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

**get_string**() → str

```
# SCPI: SYSTem:LOCK:SHARed:STRing
value: str = driver.system.lock.shared.get_string()
```

No command help available

**return**
string: No help available

### 6.21.19 MassMemory

**class MassMemoryCls**

MassMemory 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.system.massMemory.clone()
```

#### Subgroups

##### 6.21.19.1 Path

#### SCPI Commands :

```
SYSTem:MMEMory:PATH
SYSTem:MMEMory:PATH:USER
```

**class PathCls**

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

**get**(*path_type: str*) → str

```
# SCPI: SYSTem:MMEMory:PATH
value: str = driver.system.massMemory.path.get(path_type = 'abc')
```

No command help available

**param path_type**
No help available

**return**
path: No help available

**get_user**() → str

```
# SCPI: SYSTem:MMEMory:PATH:USER
value: str = driver.system.massMemory.path.get_user()
```

Queries the user directory, that means the directory the R&S SMA100B stores user files on.

**return**
path_user: string

### 6.21.20 Ntp

#### SCPI Command :

```
SYSTem:NTP:HOSTname
```

**class NtpCls**

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

**get_hostname**() → str

```
# SCPI: SYSTem:NTP:HOSTname
value: str = driver.system.ntp.get_hostname()
```

Sets the address of the NTP server. You can enter the IP address, or the hostname of the time server, or even set up an own vendor zone. See the Internet for more information on NTP.

**return**
ntp_name: string

**set_hostname**(*ntp_name: str*) → None

```
# SCPI: SYSTem:NTP:HOSTname
driver.system.ntp.set_hostname(ntp_name = 'abc')
```

Sets the address of the NTP server. You can enter the IP address, or the hostname of the time server, or even set up an own vendor zone. See the Internet for more information on NTP.

**param ntp_name**
string

### 6.21.21 Package

**class PackageCls**

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

#### Cloning the Group

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

### Subgroups

#### 6.21.21.1 ChartDisplay

### SCPI Command :

```
SYSTem:PACKage:CHARtdisplay:VERSion
```

**class ChartDisplayCls**

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

**get_version**() → str

```
# SCPI: SYSTem:PACKage:CHARtdisplay:VERSion
value: str = driver.system.package.chartDisplay.get_version()
```

No command help available

**return**
version: No help available

#### 6.21.21.2 GuiFramework

### SCPI Command :

```
SYSTem:PACKage:GUIFramework:VERSion
```

**class GuiFrameworkCls**

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

**get_version**() → str

```
# SCPI: SYSTem:PACKage:GUIFramework:VERSion
value: str = driver.system.package.guiFramework.get_version()
```

No command help available

**return**
version: No help available

#### 6.21.21.3 Qt

### SCPI Command :

```
SYSTem:PACKage:QT:VERSion
```

**class QtCls**

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

**get_version**() → str

```
# SCPI: SYSTem:PACKage:QT:VERSion
value: str = driver.system.package.qt.get_version()
```

No command help available

**return**
version: No help available

## 6.21.22 PciFpga

**class PciFpgaCls**

PciFpga commands group definition. 5 total commands, 1 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.21.22.1 Update

### SCPI Command :

```
SYSTem:PCIFpga:UPDate
```

**class UpdateCls**

Update commands group definition. 5 total commands, 3 Subgroups, 1 group commands

**set**() → None

```
# SCPI: SYSTem:PCIFpga:UPDate
driver.system.pciFpga.update.set()
```

No command help available

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

```
# SCPI: SYSTem:PCIFpga:UPDate
driver.system.pciFpga.update.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsSmab.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.system.pciFpga.update.clone()
```

### Subgroups

#### 6.21.22.1.1 Check

### SCPI Command :

```
SYSTem:PCIFpga:UPDate:CHECk
```

**class CheckCls**

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

**set**() → None

```
# SCPI: SYSTem:PCIFpga:UPDate:CHECk
driver.system.pciFpga.update.check.set()
```

No command help available

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

```
# SCPI: SYSTem:PCIFpga:UPDate:CHECk
driver.system.pciFpga.update.check.set_with_opc()
```

No command help available

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

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

#### 6.21.22.1.2 Needed

### SCPI Command :

```
SYSTem:PCIFpga:UPDate:NEEDed:[STATe]
```

**class NeededCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:PCIFpga:UPDate:NEEDed:[STATe]
value: bool = driver.system.pciFpga.update.needed.get_state()
```

No command help available

**return**
update_needed: No help available

#### 6.21.22.1.3 Tselected

**SCPI Commands :**

```
SYSTem:PCIFpga:UPDate:TSELected:CATalog
SYSTem:PCIFpga:UPDate:TSELected:STEP
```

**class TselectedCls**

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

**get_catalog**() → str

```
# SCPI: SYSTem:PCIFpga:UPDate:TSELected:CATalog
value: str = driver.system.pciFpga.update.tselected.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_step**() → str

```
# SCPI: SYSTem:PCIFpga:UPDate:TSELected:STEP
value: str = driver.system.pciFpga.update.tselected.get_step()
```

No command help available

**return**
sel_string: No help available

**set_step**(*sel_string: str*) → None

```
# SCPI: SYSTem:PCIFpga:UPDate:TSELected:STEP
driver.system.pciFpga.update.tselected.set_step(sel_string = 'abc')
```

No command help available

**param sel_string**
No help available

### 6.21.23 Profiling

**SCPI Command :**

```
SYSTem:PROFiling:STATe
```

**class ProfilingCls**

Profiling commands group definition. 18 total commands, 6 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:STATe
value: bool = driver.system.profiling.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:STATe
driver.system.profiling.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.23.1 HwAccess

### SCPI Commands :

```
SYSTem:PROFiling:HWACcess:DESCription
SYSTem:PROFiling:HWACcess:PDURation
SYSTem:PROFiling:HWACcess:STATe
```

**class HwAccessCls**

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

**get_description**() → str

```
# SCPI: SYSTem:PROFiling:HWACcess:DESCription
value: str = driver.system.profiling.hwAccess.get_description()
```

No command help available

**return**
description: No help available

**get_pduration**() → int

```
# SCPI: SYSTem:PROFiling:HWACcess:PDURation
value: int = driver.system.profiling.hwAccess.get_pduration()
```

No command help available

**return**
duration_us: No help available

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:HWACcess:STATe
value: bool = driver.system.profiling.hwAccess.get_state()
```

No command help available

**return**
state: No help available

**set_pduration**(*duration_us: int*) → None

```
# SCPI: SYSTem:PROFiling:HWACcess:PDURation
driver.system.profiling.hwAccess.set_pduration(duration_us = 1)
```

No command help available

**param duration_us**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:HWACcess:STATe
driver.system.profiling.hwAccess.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.21.23.2 Logging

##### SCPI Command :

```
SYSTem:PROFiling:LOGGing:STATe
```

**class LoggingCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:LOGGing:STATe
value: bool = driver.system.profiling.logging.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:LOGGing:STATe
driver.system.profiling.logging.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.21.23.3 Module

**SCPI Commands :**

```
SYSTem:PROFiling:MODule:CATalog
SYSTem:PROFiling:MODule:STATe
```

**class ModuleCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:PROFiling:MODule:CATalog
value: List[str] = driver.system.profiling.module.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:MODule:STATe
value: bool = driver.system.profiling.module.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:MODule:STATe
driver.system.profiling.module.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.21.23.4 Record

**SCPI Commands :**

```
SYSTem:PROFiling:RECord
SYSTem:PROFiling:RECord:CLEar
SYSTem:PROFiling:RECord:IGNore
SYSTem:PROFiling:RECord:SAVE
```

### class RecordCls

Record commands group definition. 7 total commands, 2 Subgroups, 4 group commands

**clear**() → None

```
# SCPI: SYSTem:PROFiling:RECord:CLEar
driver.system.profiling.record.clear()
```

No command help available

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

```
# SCPI: SYSTem:PROFiling:RECord:CLEar
driver.system.profiling.record.clear_with_opc()
```

No command help available

Same as clear, but waits for the operation to complete before continuing further. Use the RsSmab.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**(*index: List[str]*) → List[str]

```
# SCPI: SYSTem:PROFiling:RECord
value: List[str] = driver.system.profiling.record.get(index = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

**param index**
No help available

**return**
index: No help available

**get_ignore**() → float

```
# SCPI: SYSTem:PROFiling:RECord:IGNore
value: float = driver.system.profiling.record.get_ignore()
```

No command help available

**return**
count: No help available

**save**(*filename: str*) → None

```
# SCPI: SYSTem:PROFiling:RECord:SAVE
driver.system.profiling.record.save(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_ignore**(*count: float*) → None

```
# SCPI: SYSTem:PROFiling:RECord:IGNore
driver.system.profiling.record.set_ignore(count = 1.0)
```

No command help available

**param count**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.23.4.1 Count

### SCPI Commands :

```
SYSTem:PROFiling:RECord:COUNt:MAX
SYSTem:PROFiling:RECord:COUNt
```

**class CountCls**

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

**get_max**() → float

```
# SCPI: SYSTem:PROFiling:RECord:COUNt:MAX
value: float = driver.system.profiling.record.count.get_max()
```

No command help available

**return**
count: No help available

**get_value**() → float

```
# SCPI: SYSTem:PROFiling:RECord:COUNt
value: float = driver.system.profiling.record.count.get_value()
```

No command help available

**return**
count: No help available

**set_max**(*count: float*) → None

```
# SCPI: SYSTem:PROFiling:RECord:COUNt:MAX
driver.system.profiling.record.count.set_max(count = 1.0)
```

No command help available

**param count**
No help available

#### 6.21.23.4.2 Wrap

**SCPI Command :**

```
SYSTem:PROFiling:RECord:WRAP:STATe
```

**class WrapCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:RECord:WRAP:STATe
value: bool = driver.system.profiling.record.wrap.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:RECord:WRAP:STATe
driver.system.profiling.record.wrap.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.21.23.5 Tick

**SCPI Command :**

```
SYSTem:PROFiling:TICK
```

**class TickCls**

Tick commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_value**() → str

```
# SCPI: SYSTem:PROFiling:TICK
value: str = driver.system.profiling.tick.get_value()
```

No command help available

**return**
answer: No help available

#### Cloning the Group

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

#### Subgroups

##### 6.21.23.5.1 Enable

#### SCPI Command :

```
SYSTem:PROFiling:TICK:ENABle
```

**class EnableCls**

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

**set**() → None

```
# SCPI: SYSTem:PROFiling:TICK:ENABle
driver.system.profiling.tick.enable.set()
```

No command help available

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

```
# SCPI: SYSTem:PROFiling:TICK:ENABle
driver.system.profiling.tick.enable.set_with_opc()
```

No command help available

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

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

##### 6.21.23.6 Tpoint

#### SCPI Command :

```
SYSTem:PROFiling:TPOint:RESTart
```

**class TpointCls**

Tpoint commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_restart**() → List[str]

```
# SCPI: SYSTem:PROFiling:TPOint:RESTart
value: List[str] = driver.system.profiling.tpoint.get_restart()
```

No command help available

**return**
module_and_tp: No help available

**set_restart**(*module_and_tp: List[str]*) → None

```
# SCPI: SYSTem:PROFiling:TPOint:RESTart
driver.system.profiling.tpoint.set_restart(module_and_tp = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

**param module_and_tp**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.23.6.1 Catalog

### SCPI Command :

```
SYSTem:PROFiling:TPOint:CATalog
```

**class CatalogCls**

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

**get**(*name: str*) → List[str]

```
# SCPI: SYSTem:PROFiling:TPOint:CATalog
value: List[str] = driver.system.profiling.tpoint.catalog.get(name = 'abc')
```

No command help available

**param name**
No help available

**return**
value: No help available

## 6.21.24 Protect<Level>

### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.system.protect.repcap_level_get()
driver.system.protect.repcap_level_set(repcap.Level.Nr1)
```

**class ProtectCls**

Protect commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Level, default value after init: Level.Nr1

### Cloning the Group

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

### Subgroups

#### 6.21.24.1 State

### SCPI Command :

```
SYSTem:PROTect<CH>:[STATe]
```

**class StateCls**

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

**get**(*level=Level.Default*) → bool

```
# SCPI: SYSTem:PROTect<CH>:[STATe]
value: bool = driver.system.protect.state.get(level = repcap.Level.Default)
```

Activates and deactivates the specified protection level.

**param level**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Protect')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *key: int = None*, *level=Level.Default*) → None

```
# SCPI: SYSTem:PROTect<CH>:[STATe]
driver.system.protect.state.set(state = False, key = 1, level = repcap.Level.
↪Default)
```

Activates and deactivates the specified protection level.

**param state**
1| ON| 0| OFF

**param key**
integer The respective functions are disabled when the protection level is activated. No password is required for activation of a level. A password must be entered to deactivate the protection level. The default password for the first level is 123456. This protection level is required to unlock internal adjustments for example.

**param level**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Protect')

### 6.21.25 Reboot

#### SCPI Command :

```
SYSTem:REBoot
```

**class RebootCls**

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

**set**() → None

```
# SCPI: SYSTem:REBoot
driver.system.reboot.set()
```

Reboots the instrument including the operating system.

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

```
# SCPI: SYSTem:REBoot
driver.system.reboot.set_with_opc()
```

Reboots the instrument including the operating system.

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

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

### 6.21.26 Restart

#### SCPI Command :

```
SYSTem:RESTart
```

**class RestartCls**

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

**set**() → None

```
# SCPI: SYSTem:RESTart
driver.system.restart.set()
```

Restarts the instrument without restarting the operating system.

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

```
# SCPI: SYSTem:RESTart
driver.system.restart.set_with_opc()
```

Restarts the instrument without restarting the operating system.

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

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

### 6.21.27 Scrpt

#### SCPI Commands :

```
SYSTem:SCRPt:ARG
SYSTem:SCRPt:CMD
SYSTem:SCRPt:DATA
SYSTem:SCRPt:RUN
```

**class ScrptCls**

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

**get_arg**() → str

```
# SCPI: SYSTem:SCRPt:ARG
value: str = driver.system.scrpt.get_arg()
```

No command help available

**return**
arguments: No help available

**get_cmd**() → str

```
# SCPI: SYSTem:SCRPt:CMD
value: str = driver.system.scrpt.get_cmd()
```

No command help available

**return**
cmd_file: No help available

**get_data**() → str

```
# SCPI: SYSTem:SCRPt:DATA
value: str = driver.system.scrpt.get_data()
```

No command help available

**return**
data_file: No help available

**run**() → None

```
# SCPI: SYSTem:SCRPt:RUN
driver.system.scrpt.run()
```

No command help available

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

```
# SCPI: SYSTem:SCRPt:RUN
driver.system.scrpt.run_with_opc()
```

No command help available

Same as run, but waits for the operation to complete before continuing further. Use the RsSmab.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_arg**(*arguments: str*) → None

```
# SCPI: SYSTem:SCRPt:ARG
driver.system.scrpt.set_arg(arguments = 'abc')
```

No command help available

**param arguments**
No help available

**set_cmd**(*cmd_file: str*) → None

```
# SCPI: SYSTem:SCRPt:CMD
driver.system.scrpt.set_cmd(cmd_file = 'abc')
```

No command help available

**param cmd_file**
No help available

**set_data**(*data_file: str*) → None

```
# SCPI: SYSTem:SCRPt:DATA
driver.system.scrpt.set_data(data_file = 'abc')
```

No command help available

**param data_file**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.27.1 Discard

### SCPI Command :

```
SYSTem:SCRPt:DISCard
```

**class DiscardCls**

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

**set**() → None

```
# SCPI: SYSTem:SCRPt:DISCard
driver.system.scrpt.discard.set()
```

No command help available

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

```
# SCPI: SYSTem:SCRPt:DISCard
driver.system.scrpt.discard.set_with_opc()
```

No command help available

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

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

### 6.21.28 Security

#### SCPI Command :

```
SYSTem:SECurity:[STATe]
```

**class SecurityCls**

Security commands group definition. 18 total commands, 6 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: SYSTem:SECurity:[STATe]
value: bool = driver.system.security.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:SECurity:[STATe]
driver.system.security.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.28.1 Mmem

**class MmemCls**

Mmem 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.system.security.mmem.clone()
```

### Subgroups

#### 6.21.28.1.1 Protect

**class ProtectCls**

Protect 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.system.security.mmem.protect.clone()
```

### Subgroups

#### 6.21.28.1.1.1 State

### SCPI Command :

```
SYSTem:SECurity:MMEM:PROTect:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:MMEM:PROTect:[STATe]
value: bool = driver.system.security.mmem.protect.state.get()
```

No command help available

**return**
mmem_prot_state: No help available

**set**(*sec_pass_word: str*, *mmem_prot_state: bool*) → None

```
# SCPI: SYSTem:SECurity:MMEM:PROTect:[STATe]
driver.system.security.mmem.protect.state.set(sec_pass_word = 'abc', mmem_prot_
↪state = False)
```

No command help available

**param sec_pass_word**
No help available

**param mmem_prot_state**
No help available

#### 6.21.28.2 Network

**class NetworkCls**

Network commands group definition. 12 total commands, 12 Subgroups, 0 group commands

##### Cloning the Group

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

##### Subgroups

#### 6.21.28.2.1 Avahi

**class AvahiCls**

Avahi 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.system.security.network.avahi.clone()
```

##### Subgroups

#### 6.21.28.2.1.1 State

##### SCPI Command :

```
SYSTem:SECurity:NETWork:AVAHi:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:AVAHi:[STATe]
value: bool = driver.system.security.network.avahi.state.get()
```

Disables the Avahi service for automatic configuration of the instrument in a network.

**return**
avahi_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *avahi_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:AVAHi:[STATe]
driver.system.security.network.avahi.state.set(sec_pass_word = 'abc', avahi_
↪state = False)
```

Disables the Avahi service for automatic configuration of the instrument in a network.

**param sec_pass_word**
string Current security password.

**param avahi_state**
1| ON| 0| OFF

#### 6.21.28.2.2 Ftp

**class FtpCls**

Ftp 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.system.security.network.ftp.clone()
```

### Subgroups

#### 6.21.28.2.2.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:FTP:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:FTP:[STATe]
value: bool = driver.system.security.network.ftp.state.get()
```

Disables FTP protocol for file transfer between the instrument and host.

**return**
ftp_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *ftp_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:FTP:[STATe]
driver.system.security.network.ftp.state.set(sec_pass_word = 'abc', ftp_state =␣
↪False)
```

Disables FTP protocol for file transfer between the instrument and host.

**param sec_pass_word**
string Current security password.

**param ftp_state**
1| ON| 0| OFF

#### 6.21.28.2.3 Http

**class HttpCls**

Http 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.system.security.network.http.clone()
```

### Subgroups

#### 6.21.28.2.3.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:HTTP:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:HTTP:[STATe]
value: bool = driver.system.security.network.http.state.get()
```

Disables control of the instrument over HTTP, the protocol for hypermedia information systems.

**return**
http_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *http_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:HTTP:[STATe]
driver.system.security.network.http.state.set(sec_pass_word = 'abc', http_state␣
↪= False)
```

Disables control of the instrument over HTTP, the protocol for hypermedia information systems.

**param sec_pass_word**
string Current security password.

**param http_state**
1| ON| 0| OFF

#### 6.21.28.2.4 Raw

**class RawCls**

Raw 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.system.security.network.raw.clone()
```

### Subgroups

#### 6.21.28.2.4.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:RAW:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:RAW:[STATe]
value: bool = driver.system.security.network.raw.state.get()
```

Disables the LAN interface for remote control of the instrument over raw socket port.

**return**
raw_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *raw_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:RAW:[STATe]
driver.system.security.network.raw.state.set(sec_pass_word = 'abc', raw_state =␣
↪False)
```

Disables the LAN interface for remote control of the instrument over raw socket port.

**param sec_pass_word**
string Current security password.

**param raw_state**
1| ON| 0| OFF

#### 6.21.28.2.5 RemSupport

**SCPI Command :**

```
SYSTem:SECurity:NETWork:REMSupport:[STATe]
```

**class RemSupportCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:REMSupport:[STATe]
value: bool = driver.system.security.network.remSupport.get_state()
```

Disables communication over SSH (SCP) for service purposes.

**return**
net_rem_support: 1| ON| 0| OFF

**set_state**(*net_rem_support: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:REMSupport:[STATe]
driver.system.security.network.remSupport.set_state(net_rem_support = False)
```

Disables communication over SSH (SCP) for service purposes.

**param net_rem_support**
1| ON| 0| OFF

#### 6.21.28.2.6 Rpc

**class RpcCls**

Rpc 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.system.security.network.rpc.clone()
```

### Subgroups

#### 6.21.28.2.6.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:RPC:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:RPC:[STATe]
value: bool = driver.system.security.network.rpc.state.get()
```

Enables the LAN interface for remote control of the instrument via remote procedure calls (RPC) .

**return**
rpc_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *rpc_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:RPC:[STATe]
driver.system.security.network.rpc.state.set(sec_pass_word = 'abc', rpc_state =␣
↪False)
```

Enables the LAN interface for remote control of the instrument via remote procedure calls (RPC) .

**param sec_pass_word**
string Current security password.

**param rpc_state**
1| ON| 0| OFF

#### 6.21.28.2.7 Smb

**class SmbCls**

Smb 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.system.security.network.smb.clone()
```

### Subgroups

#### 6.21.28.2.7.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:SMB:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SMB:[STATe]
value: bool = driver.system.security.network.smb.state.get()
```

Disables access to the file system, printers and serial ports in a network over SMB.

**return**
smb_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *smb_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SMB:[STATe]
driver.system.security.network.smb.state.set(sec_pass_word = 'abc', smb_state =␣
↪False)
```

Disables access to the file system, printers and serial ports in a network over SMB.

**param sec_pass_word**
string Current security password.

**param smb_state**
1| ON| 0| OFF

#### 6.21.28.2.8 Soe

**class SoeCls**

Soe 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.system.security.network.soe.clone()
```

### Subgroups

#### 6.21.28.2.8.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:SOE:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SOE:[STATe]
value: bool = driver.system.security.network.soe.state.get()
```

Disables control of the instrument over LAN using SCPI commands.

**return**
soe_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *soe_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SOE:[STATe]
driver.system.security.network.soe.state.set(sec_pass_word = 'abc', soe_state =␣
↪False)
```

Disables control of the instrument over LAN using SCPI commands.

**param sec_pass_word**
string Current security password.

**param soe_state**
1| ON| 0| OFF

#### 6.21.28.2.9 Ssh

**class SshCls**

Ssh 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.system.security.network.ssh.clone()
```

#### Subgroups

##### 6.21.28.2.9.1 State

#### SCPI Command :

```
SYSTem:SECurity:NETWork:SSH:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SSH:[STATe]
value: bool = driver.system.security.network.ssh.state.get()
```

Disables control of the instrument over LAN using the SSH network protocol.

**return**
ssh_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *ssh_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SSH:[STATe]
driver.system.security.network.ssh.state.set(sec_pass_word = 'abc', ssh_state =␣
↪False)
```

Disables control of the instrument over LAN using the SSH network protocol.

**param sec_pass_word**
string Current security password.

**param ssh_state**
1| ON| 0| OFF

##### 6.21.28.2.10 State

#### SCPI Command :

```
SYSTem:SECurity:NETWork:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:[STATe]
value: bool = driver.system.security.network.state.get()
```

Disables the LAN interface in general, including all services.

**return**
lan_stor_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *lan_stor_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:[STATe]
driver.system.security.network.state.set(sec_pass_word = 'abc', lan_stor_state
↪= False)
```

Disables the LAN interface in general, including all services.

**param sec_pass_word**
string Current security password. The default password is 123456.

**param lan_stor_state**
1| ON| 0| OFF

#### 6.21.28.2.11 SwUpdate

**class SwUpdateCls**

SwUpdate 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.system.security.network.swUpdate.clone()
```

### Subgroups

#### 6.21.28.2.11.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:SWUPdate:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SWUPdate:[STATe]
value: bool = driver.system.security.network.swUpdate.state.get()
```

Disables software update over LAN.

**return**
sw_update_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *sw_update_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SWUPdate:[STATe]
driver.system.security.network.swUpdate.state.set(sec_pass_word = 'abc', sw_
↪update_state = False)
```

Disables software update over LAN.

**param sec_pass_word**
string Current security password.

**param sw_update_state**
1| ON| 0| OFF

#### 6.21.28.2.12 Vnc

**class VncCls**

Vnc 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.system.security.network.vnc.clone()
```

##### Subgroups

###### 6.21.28.2.12.1 State

##### SCPI Command :

```
SYSTem:SECurity:NETWork:VNC:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:VNC:[STATe]
value: bool = driver.system.security.network.vnc.state.get()
```

Disables the VNC interface for remote control of the instrument.

**return**
vnc_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *vnc_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:VNC:[STATe]
driver.system.security.network.vnc.state.set(sec_pass_word = 'abc', vnc_state =␣
↪False)
```

Disables the VNC interface for remote control of the instrument.

**param sec_pass_word**
string Current security password.

**param vnc_state**
1| ON| 0| OFF

#### 6.21.28.3 Sanitize

**class SanitizeCls**

Sanitize 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.system.security.sanitize.clone()
```

### Subgroups

#### 6.21.28.3.1 State

### SCPI Command :

```
SYSTem:SECurity:SANitize:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:SANitize:[STATe]
value: bool = driver.system.security.sanitize.state.get()
```

Sanitizes the internal memory.

**return**
mmem_prot_state: 0| 1| OFF| ON

**set**(*sec_pass_word: str*, *mmem_prot_state: bool*) → None

```
# SCPI: SYSTem:SECurity:SANitize:[STATe]
driver.system.security.sanitize.state.set(sec_pass_word = 'abc', mmem_prot_
↪state = False)
```

Sanitizes the internal memory.

**param sec_pass_word**
string

**param mmem_prot_state**
0| 1| OFF| ON

#### 6.21.28.4 SuPolicy

**SCPI Command :**

```
SYSTem:SECurity:SUPolicy
```

**class SuPolicyCls**

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

**get**() → UpdPolicyMode

```
# SCPI: SYSTem:SECurity:SUPolicy
value: enums.UpdPolicyMode = driver.system.security.suPolicy.get()
```

Configures the automatic signature verification for firmware installation.

**return**
update_policy: STRict| CONFirm| IGNore

**set**(*sec_pass_word: str*, *update_policy: UpdPolicyMode*) → None

```
# SCPI: SYSTem:SECurity:SUPolicy
driver.system.security.suPolicy.set(sec_pass_word = 'abc', update_policy =␣
↪enums.UpdPolicyMode.CONFirm)
```

Configures the automatic signature verification for firmware installation.

**param sec_pass_word**
string

**param update_policy**
STRict| CONFirm| IGNore

#### 6.21.28.5 UsbStorage

**class UsbStorageCls**

UsbStorage 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.system.security.usbStorage.clone()
```

**Subgroups**

##### 6.21.28.5.1 State

**SCPI Command :**

```
SYSTem:SECurity:USBStorage:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:USBStorage:[STATe]
value: bool = driver.system.security.usbStorage.state.get()
```

No command help available

**return**
usb_stor_state: No help available

**set**(*sec_pass_word: str*, *usb_stor_state: bool*) → None

```
# SCPI: SYSTem:SECurity:USBStorage:[STATe]
driver.system.security.usbStorage.state.set(sec_pass_word = 'abc', usb_stor_
↪state = False)
```

No command help available

**param sec_pass_word**
No help available

**param usb_stor_state**
No help available

#### 6.21.28.6 VolMode

**class VolModeCls**

VolMode 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.system.security.volMode.clone()
```

##### Subgroups

###### 6.21.28.6.1 State

##### SCPI Command :

```
SYSTem:SECurity:VOLMode:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:VOLMode:[STATe]
value: bool = driver.system.security.volMode.state.get()
```

Activates volatile mode, so that no user data can be written to the internal memory permanently. To enable volatile mode, reboot the instrument. Otherwise the change has no effect.

**return**
mmem_prot_state: 0| 1| OFF| ON

**set**(*sec_pass_word: str*, *mmem_prot_state: bool*) → None

```
# SCPI: SYSTem:SECurity:VOLMode:[STATe]
driver.system.security.volMode.state.set(sec_pass_word = 'abc', mmem_prot_state␣
↪= False)
```

Activates volatile mode, so that no user data can be written to the internal memory permanently. To enable volatile mode, reboot the instrument. Otherwise the change has no effect.

**param sec_pass_word**
string Current security password The default password is 123456.

**param mmem_prot_state**
0| 1| OFF| ON

### 6.21.29 Shutdown

#### SCPI Command :

```
SYSTem:SHUTdown
```

**class ShutdownCls**

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

**set**() → None

```
# SCPI: SYSTem:SHUTdown
driver.system.shutdown.set()
```

Shuts down the instrument.

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

```
# SCPI: SYSTem:SHUTdown
driver.system.shutdown.set_with_opc()
```

Shuts down the instrument.

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

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

## 6.21.30 Specification

**class SpecificationCls**

Specification commands group definition. 5 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.21.30.1 Identification

#### SCPI Command :

```
SYSTem:SPECification:IDENtification:CATalog
```

**class IdentificationCls**

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

**get_catalog**() → str

```
# SCPI: SYSTem:SPECification:IDENtification:CATalog
value: str = driver.system.specification.identification.get_catalog()
```

Queries the parameter identifiers (<Id>) available in the data sheet.

**return**
id_list: string Comma-separated string of the parameter identifiers (Id)

#### 6.21.30.2 Version

#### SCPI Commands :

```
SYSTem:SPECification:VERSion:CATalog
SYSTem:SPECification:VERSion:FACTory
SYSTem:SPECification:VERSion:SFACtory
SYSTem:SPECification:VERSion
```

**class VersionCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:SPECification:VERSion:CATalog
value: List[str] = driver.system.specification.version.get_catalog()
```

Queries all data sheet versions stored in the instrument.

**return**
vers_catalog: string

**get_factory**() → str

```
# SCPI: SYSTem:SPECification:VERSion:FACTory
value: str = driver.system.specification.version.get_factory()
```

Queries the data sheet version of the factory setting.

**return**
version: string

**get_sfactory**() → str

```
# SCPI: SYSTem:SPECification:VERSion:SFACtory
value: str = driver.system.specification.version.get_sfactory()
```

No command help available

**return**
ds_fact_version: No help available

**get_value**() → str

```
# SCPI: SYSTem:SPECification:VERSion
value: str = driver.system.specification.version.get_value()
```

Selects a data sheet version from the data sheets saved on the instrument. Further queries regarding the data sheet parameters (<Id>) and their values refer to the selected data sheet. To query the list of data sheet versions, use the command method RsSmab.System.Specification.Version.catalog.

**return**
version: string

**set_sfactory**(*ds_fact_version: str*) → None

```
# SCPI: SYSTem:SPECification:VERSion:SFACtory
driver.system.specification.version.set_sfactory(ds_fact_version = 'abc')
```

No command help available

**param ds_fact_version**
No help available

**set_value**(*version: str*) → None

```
# SCPI: SYSTem:SPECification:VERSion
driver.system.specification.version.set_value(version = 'abc')
```

Selects a data sheet version from the data sheets saved on the instrument. Further queries regarding the data sheet parameters (<Id>) and their values refer to the selected data sheet. To query the list of data sheet versions, use the command method RsSmab.System.Specification.Version.catalog.

**param version**
string

### 6.21.31 SrData

#### SCPI Commands :

```
SYSTem:SRData:DELete
SYSTem:SRData
```

**class SrDataCls**

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

**delete**() → None

```
# SCPI: SYSTem:SRData:DELete
driver.system.srData.delete()
```

No command help available

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

```
# SCPI: SYSTem:SRData:DELete
driver.system.srData.delete_with_opc()
```

No command help available

Same as delete, but waits for the operation to complete before continuing further. Use the RsSmab.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**() → bytes

```
# SCPI: SYSTem:SRData
value: bytes = driver.system.srData.get_value()
```

Queris the SCPI recording data from the internal file. This feature enables you to transfer an instrument configuration to other test environments, as e.g. laboratory virtual instruments.

**return**
file_data: block data

### 6.21.32 Srexec

#### SCPI Command :

```
SYSTem:SREXec
```

**class SrexecCls**

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

**set**() → None

```
# SCPI: SYSTem:SREXec
driver.system.srexec.set()
```

No command help available

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

```
# SCPI: SYSTem:SREXec
driver.system.srexec.set_with_opc()
```

No command help available

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

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

### 6.21.33 Srtime

#### SCPI Command :

```
SYSTem:SRTime:STATe
```

**class SrtimeCls**

Srtime commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: SYSTem:SRTime:STATe
value: bool = driver.system.srtime.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:SRTime:STATe
driver.system.srtime.set_state(state = False)
```

No command help available

**param state**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.21.33.1 Synchronize

#### SCPI Command :

```
SYSTem:SRTime:SYNChronize
```

**class SynchronizeCls**

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

**get**(*time: str*) → str

```
# SCPI: SYSTem:SRTime:SYNChronize
value: str = driver.system.srtime.synchronize.get(time = 'abc')
```

No command help available

**param time**
No help available

**return**
time: No help available

### 6.21.34 Startup

#### SCPI Command :

```
SYSTem:STARtup:COMPlete
```

**class StartupCls**

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

**get_complete**() → bool

```
# SCPI: SYSTem:STARtup:COMPlete
value: bool = driver.system.startup.get_complete()
```

Queries if the startup of the instrument is completed.

**return**
complete: 1| ON| 0| OFF

### 6.21.35 Time

#### SCPI Commands :

```
SYSTem:TIME
SYSTem:TIME:LOCal
SYSTem:TIME:PROTocol
SYSTem:TIME:UTC
```

**class TimeCls**

Time commands group definition. 12 total commands, 3 Subgroups, 4 group commands

**class TimeStruct**

Response structure. Fields:

- Hour: List[int]: integer Range: 0 to 23
- Minute: int: integer Range: 0 to 59
- Second: int: integer Range: 0 to 59

**get**() → TimeStruct

```
# SCPI: SYSTem:TIME
value: TimeStruct = driver.system.time.get()
```

Queries or sets the time for the instrument-internal clock. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State.set.

**return**
structure: for return value, see the help for TimeStruct structure arguments.

**get_local**() → str

```
# SCPI: SYSTem:TIME:LOCal
value: str = driver.system.time.get_local()
```

No command help available

**return**
pseudo_string: No help available

**get_protocol**() → TimeProtocol

```
# SCPI: SYSTem:TIME:PROTocol
value: enums.TimeProtocol = driver.system.time.get_protocol()
```

Sets the date and time of the operating system.

**return**
time_protocol: OFF| NONE| 0| NTP| ON| 1 NONE Sets the date and time according to the selected timezone, see method RsSmab.System.Time.Zone.catalog and method RsSmab.System.Time.Zone.value. NTP Sets the date and time derived from the network time protocol. To select the NTP time server, use the commands method RsSmab.System.Ntp.hostname and SYSTem:NTP:STATe.

**get_utc**() → str

```
# SCPI: SYSTem:TIME:UTC
value: str = driver.system.time.get_utc()
```

No command help available

**return**
pseudo_string: No help available

**set**(*hour: List[int]*, *minute: int*, *second: int*) → None

```
# SCPI: SYSTem:TIME
driver.system.time.set(hour = [1, 2, 3], minute = 1, second = 1)
```

Queries or sets the time for the instrument-internal clock. This is a password-protected function. Unlock the protection level 1 to access it. See method RsSmab.System.Protect.State.set.

**param hour**
integer Range: 0 to 23

**param minute**
integer Range: 0 to 59

**param second**
integer Range: 0 to 59

**set_local**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:LOCal
driver.system.time.set_local(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**set_protocol**(*time_protocol: TimeProtocol*) → None

```
# SCPI: SYSTem:TIME:PROTocol
driver.system.time.set_protocol(time_protocol = enums.TimeProtocol._0)
```

Sets the date and time of the operating system.

**param time_protocol**
OFF| NONE| 0| NTP| ON| 1 NONE Sets the date and time according to the selected timezone, see method RsSmab.System.Time.Zone.catalog and method RsSmab.System.Time.Zone.value. NTP Sets the date and time derived from the network time protocol. To select the NTP time server, use the commands method RsSmab.System.Ntp.hostname and SYSTem:NTP:STATe.

**set_utc**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:UTC
driver.system.time.set_utc(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.35.1 DaylightSavingTime

### SCPI Command :

```
SYSTem:TIME:DSTime:MODE
```

**class DaylightSavingTimeCls**

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

**get_mode**() → str

```
# SCPI: SYSTem:TIME:DSTime:MODE
value: str = driver.system.time.daylightSavingTime.get_mode()
```

No command help available

**return**
pseudo_string: No help available

**set_mode**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:DSTime:MODE
driver.system.time.daylightSavingTime.set_mode(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.35.1.1 Rule

### SCPI Commands :

```
SYSTem:TIME:DSTime:RULE:CATalog
SYSTem:TIME:DSTime:RULE
```

**class RuleCls**

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

**get_catalog**() → str

```
# SCPI: SYSTem:TIME:DSTime:RULE:CATalog
value: str = driver.system.time.daylightSavingTime.rule.get_catalog()
```

No command help available

**return**
pseudo_string: No help available

**get_value**() → str

```
# SCPI: SYSTem:TIME:DSTime:RULE
value: str = driver.system.time.daylightSavingTime.rule.get_value()
```

No command help available

**return**
pseudo_string: No help available

**set_value**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:DSTime:RULE
driver.system.time.daylightSavingTime.rule.set_value(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

#### 6.21.35.2 HrTimer

**SCPI Command :**

```
SYSTem:TIME:HRTimer:RELative
```

**class HrTimerCls**

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

**set_relative**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:HRTimer:RELative
driver.system.time.hrTimer.set_relative(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.35.2.1 Absolute

### SCPI Commands :

```
SYSTem:TIME:HRTimer:ABSolute:SET
SYSTem:TIME:HRTimer:ABSolute
```

**class AbsoluteCls**

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

**get_set**() → str

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:SET
value: str = driver.system.time.hrTimer.absolute.get_set()
```

No command help available

**return**
pseudo_string: No help available

**set_set**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:SET
driver.system.time.hrTimer.absolute.set_set(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**set_value**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute
driver.system.time.hrTimer.absolute.set_value(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

#### 6.21.35.3 Zone

**SCPI Commands :**

```
SYSTem:TIME:ZONE:CATalog
SYSTem:TIME:ZONE
```

**class ZoneCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:TIME:ZONE:CATalog
value: List[str] = driver.system.time.zone.get_catalog()
```

Querys the list of available timezones.

**return**
catalog: No help available

**get_value**() → str

```
# SCPI: SYSTem:TIME:ZONE
value: str = driver.system.time.zone.get_value()
```

Sets the timezone. You can query the list of the available timezones with method RsSmab.System.Time.Zone.catalog.

**return**
time_zone: string

**set_value**(*time_zone: str*) → None

```
# SCPI: SYSTem:TIME:ZONE
driver.system.time.zone.set_value(time_zone = 'abc')
```

Sets the timezone. You can query the list of the available timezones with method RsSmab.System.Time.Zone.catalog.

**param time_zone**
string

### 6.21.36 Ulock

**SCPI Command :**

```
SYSTem:ULOCk
```

**class UlockCls**

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

**get**() → DispKeybLockMode

```
# SCPI: SYSTem:ULOCk
value: enums.DispKeybLockMode = driver.system.ulock.get()
```

Locks or unlocks the user interface of the instrument.

**return**
mode: ENABled| DONLy| DISabled| TOFF| VNConly ENABled Unlocks the display, the touchscreen and all controls for the manual operation. DONLy Locks the touchscreen and controls for the manual operation of the instrument. The display shows the current settings. VNConly Locks the touchscreen and controls for the manual operation, and enables remote operation over VNC. The display shows the current settings. TOFF Locks the touchscreen for the manual operation of the instrument. The display shows the current settings. DISabled Locks the display, the touchscreen and all controls for the manual operation.

**set**(*sec_pass_word: str*, *mode: DispKeybLockMode*) → None

```
# SCPI: SYSTem:ULOCk
driver.system.ulock.set(sec_pass_word = 'abc', mode = enums.DispKeybLockMode.
↪DISabled)
```

Locks or unlocks the user interface of the instrument.

**param sec_pass_word**
No help available

**param mode**
ENABled| DONLy| DISabled| TOFF| VNConly ENABled Unlocks the display, the touchscreen and all controls for the manual operation. DONLy Locks the touchscreen and controls for the manual operation of the instrument. The display shows the current settings. VNConly Locks the touchscreen and controls for the manual operation, and enables remote operation over VNC. The display shows the current settings. TOFF Locks the touchscreen for the manual operation of the instrument. The display shows the current settings. DISabled Locks the display, the touchscreen and all controls for the manual operation.

### 6.21.37 Undo

#### SCPI Command :

```
SYSTem:UNDO:STATe
```

**class UndoCls**

Undo commands group definition. 5 total commands, 3 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: SYSTem:UNDO:STATe
value: bool = driver.system.undo.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:UNDO:STATe
driver.system.undo.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.21.37.1 Hclear

### SCPI Command :

```
SYSTem:UNDO:HCLear
```

**class HclearCls**

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

**set**() → None

```
# SCPI: SYSTem:UNDO:HCLear
driver.system.undo.hclear.set()
```

No command help available

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

```
# SCPI: SYSTem:UNDO:HCLear
driver.system.undo.hclear.set_with_opc()
```

No command help available

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

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

#### 6.21.37.2 Hid

### SCPI Command :

```
SYSTem:UNDO:HID:SELect
```

**class HidCls**

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

**set_select**(*select: int*) → None

```
# SCPI: SYSTem:UNDO:HID:SELect
driver.system.undo.hid.set_select(select = 1)
```

No command help available

**param select**
No help available

#### 6.21.37.3 Hlable

**SCPI Commands :**

```
SYSTem:UNDO:HLABle:CATalog
SYSTem:UNDO:HLABle:SELect
```

**class HlableCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:UNDO:HLABle:CATalog
value: List[str] = driver.system.undo.hlable.get_catalog()
```

No command help available

**return**
catalog: No help available

**set_select**(*label: str*) → None

```
# SCPI: SYSTem:UNDO:HLABle:SELect
driver.system.undo.hlable.set_select(label = 'abc')
```

No command help available

**param label**
No help available

## 6.22 Test

**SCPI Commands :**

```
TEST:LEVel
TEST:NRPTrigger
TEST:PRESet
```

**class TestCls**

Test commands group definition. 20 total commands, 8 Subgroups, 3 group commands

**get_level**() → SelftLev

```
# SCPI: TEST:LEVel
value: enums.SelftLev = driver.test.get_level()
```

No command help available

**return**
level: No help available

**preset**() → None

```
# SCPI: TEST:PRESet
driver.test.preset()
```

No command help available

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

```
# SCPI: TEST:PRESet
driver.test.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmab.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_level**(*level: SelftLev*) → None

```
# SCPI: TEST:LEVel
driver.test.set_level(level = enums.SelftLev.CUSTomer)
```

No command help available

**param level**
No help available

**set_nrp_trigger**(*nrp_trigger: bool*) → None

```
# SCPI: TEST:NRPTrigger
driver.test.set_nrp_trigger(nrp_trigger = False)
```

No command help available

**param nrp_trigger**
No help available

#### Cloning the Group

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

#### Subgroups

### 6.22.1 All

#### SCPI Commands :

```
TEST<HW>:ALL:RESult
TEST<HW>:ALL:STARt
```

**class AllCls**

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

**get_result**() → Test

```
# SCPI: TEST<HW>:ALL:RESult
value: enums.Test = driver.test.all.get_result()
```

Queries the result of the performed selftest. Start the selftest with method RsSmab.Test.All.start.

**return**
result: 0| 1| RUNning| STOPped

**start**() → None

```
# SCPI: TEST<HW>:ALL:STARt
driver.test.all.start()
```

No command help available

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

```
# SCPI: TEST<HW>:ALL:STARt
driver.test.all.start_with_opc()
```

No command help available

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

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

### 6.22.2 Device

**class DeviceCls**

Device 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.test.device.clone()
```

#### Subgroups

##### 6.22.2.1 Internal

#### SCPI Command :

```
TEST:DEVice:INTernal
```

**class InternalCls**

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

**get**(*argument: str*) → Test

```
# SCPI: TEST:DEVice:INTernal
value: enums.Test = driver.test.device.internal.get(argument = 'abc')
```

No command help available

**param argument**
No help available

**return**
result: No help available

### 6.22.3 Pixel

#### SCPI Commands :

```
TEST:PIXel:COLor
TEST:PIXel:GRADient
TEST:PIXel:POINtsize
TEST:PIXel:RGBA
TEST:PIXel:TEXT
TEST:PIXel:WINDow
```

**class PixelCls**

Pixel commands group definition. 6 total commands, 0 Subgroups, 6 group commands

**get_gradient**() → bool

```
# SCPI: TEST:PIXel:GRADient
value: bool = driver.test.pixel.get_gradient()
```

No command help available

**return**
pix_test_grad_stat: No help available

**get_point_size**() → int

```
# SCPI: TEST:PIXel:POINtsize
value: int = driver.test.pixel.get_point_size()
```

No command help available

**return**
pix_test_grad_stat: No help available

**get_rgba**() → List[int]

```
# SCPI: TEST:PIXel:RGBA
value: List[int] = driver.test.pixel.get_rgba()
```

No command help available

**return**
pixel_test_rgba: No help available

**get_text**() → bool

```
# SCPI: TEST:PIXel:TEXT
value: bool = driver.test.pixel.get_text()
```

No command help available

**return**
pix_test_grad_stat: No help available

**set_color**(*pix_test_color: PixelTestPredefined*) → None

```
# SCPI: TEST:PIXel:COLor
driver.test.pixel.set_color(pix_test_color = enums.PixelTestPredefined.AUTO)
```

No command help available

**param pix_test_color**
No help available

**set_gradient**(*pix_test_grad_stat: bool*) → None

```
# SCPI: TEST:PIXel:GRADient
driver.test.pixel.set_gradient(pix_test_grad_stat = False)
```

No command help available

**param pix_test_grad_stat**
No help available

**set_point_size**(*pix_test_grad_stat: int*) → None

```
# SCPI: TEST:PIXel:POINtsize
driver.test.pixel.set_point_size(pix_test_grad_stat = 1)
```

No command help available

**param pix_test_grad_stat**
No help available

**set_rgba**(*pixel_test_rgba: List[int]*) → None

```
# SCPI: TEST:PIXel:RGBA
driver.test.pixel.set_rgba(pixel_test_rgba = [1, 2, 3])
```

No command help available

**param pixel_test_rgba**
No help available

**set_text**(*pix_test_grad_stat: bool*) → None

```
# SCPI: TEST:PIXel:TEXT
driver.test.pixel.set_text(pix_test_grad_stat = False)
```

No command help available

**param pix_test_grad_stat**
No help available

**set_window**(*pix_test_window: bool*) → None

```
# SCPI: TEST:PIXel:WINDow
driver.test.pixel.set_window(pix_test_window = False)
```

No command help available

**param pix_test_window**
No help available

### 6.22.4 Remote

**class RemoteCls**

Remote 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.test.remote.clone()
```

#### Subgroups

##### 6.22.4.1 Lockout

#### SCPI Command :

```
TEST<HW>:REMote:LOCKout:[STATe]
```

**class LockoutCls**

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

**set_state**(*state: bool*) → None

```
# SCPI: TEST<HW>:REMote:LOCKout:[STATe]
driver.test.remote.lockout.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.22.5 Res

#### SCPI Commands :

```
TEST:RES:COLor
TEST:RES:TEXT
TEST:RES:WIND
```

**class ResCls**

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

**get_color**() → Colour

```
# SCPI: TEST:RES:COLor
value: enums.Colour = driver.test.res.get_color()
```

No command help available

**return**
color: No help available

**get_text**() → str

```
# SCPI: TEST:RES:TEXT
value: str = driver.test.res.get_text()
```

No command help available

**return**
text: No help available

**get_wind**() → bool

```
# SCPI: TEST:RES:WIND
value: bool = driver.test.res.get_wind()
```

No command help available

**return**
state: No help available

**set_color**(*color: Colour*) → None

```
# SCPI: TEST:RES:COLor
driver.test.res.set_color(color = enums.Colour.GREen)
```

No command help available

**param color**
No help available

**set_text**(*text: str*) → None

```
# SCPI: TEST:RES:TEXT
driver.test.res.set_text(text = 'abc')
```

No command help available

**param text**
No help available

**set_wind**(*state: bool*) → None

```
# SCPI: TEST:RES:WIND
driver.test.res.set_wind(state = False)
```

No command help available

**param state**
No help available

### 6.22.6 Serror

#### SCPI Command :

```
TEST:SERRor:UNSet
```

**class SerrorCls**

Serror commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**set_unset**(*path: int*) → None

```
# SCPI: TEST:SERRor:UNSet
driver.test.serror.set_unset(path = 1)
```

No command help available

**param path**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.22.6.1 Set

#### SCPI Command :

```
TEST:SERRor:SET
```

**class SetCls**

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

**set**(*err_code: int*, *path: int*) → None

```
# SCPI: TEST:SERRor:SET
driver.test.serror.set.set(err_code = 1, path = 1)
```

No command help available

**param err_code**
No help available

**param path**
No help available

### 6.22.7 Sw

**class SwCls**

Sw 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.test.sw.clone()
```

#### Subgroups

##### 6.22.7.1 Scmd

#### SCPI Command :

```
TEST<HW>:SW:SCMD
```

**class ScmdCls**

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

**class ScmdStruct**

Response structure. Fields:

- Scmd: str: No parameter help available
- What_Is_This: str: No parameter help available

**get**() → ScmdStruct

```
# SCPI: TEST<HW>:SW:SCMD
value: ScmdStruct = driver.test.sw.scmd.get()
```

No command help available

**return**
structure: for return value, see the help for ScmdStruct structure arguments.

**set**(*scmd: str*, *what_is_this: str*) → None

```
# SCPI: TEST<HW>:SW:SCMD
driver.test.sw.scmd.set(scmd = 'abc', what_is_this = 'abc')
```

No command help available

**param scmd**
No help available

**param what_is_this**
No help available

### 6.22.8 Write

#### SCPI Command :

```
TEST:WRITe:RESult
```

**class WriteCls**

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

**set_result**(*result: SelftLevWrite*) → None

```
# SCPI: TEST:WRITe:RESult
driver.test.write.set_result(result = enums.SelftLevWrite.CUSTomer)
```

No command help available

**param result**
No help available

## 6.23 Trace<Trace>

### RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.trace.repcap_trace_get()
driver.trace.repcap_trace_set(repcap.Trace.Nr1)
```

**class TraceCls**

Trace commands group definition. 58 total commands, 4 Subgroups, 0 group commands Repeated Capability: Trace, default value after init: Trace.Nr1

### Cloning the Group

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

### Subgroups

## 6.23.1 Freq

**class FreqCls**

Freq 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.trace.freq.clone()
```

### Subgroups

#### 6.23.1.1 Sweep

**class SweepCls**

Sweep 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.trace.freq.sweep.clone()
```

### Subgroups

#### 6.23.1.1.1 Src

### SCPI Command :

```
TRACe<CH>:FREQ:SWEep:SRC
```

**class SrcCls**

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

**get**(*trace=Trace.Default*) → TraceSourceAll

```
# SCPI: TRACe<CH>:FREQ:SWEep:SRC
value: enums.TraceSourceAll = driver.trace.freq.sweep.src.get(trace = repcap.
↪Trace.Default)
```

Determines the source of a trace for display in frequency measurement mode.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
freq_source: OFF| SEN1| SEN2| SEN3| SEN4| HOLD| REF| ON ON|OFF Activates ofr deactivates the display of a trace. SEN1|SEN2|SEN3|SEN4 Activates the measurement results display of the sensor that is assigned to the trace. REF Selects a reference trace. HOLD Freezes the measurement results display of the sensor that is assigned to the trace.

**set**(*freq_source: TraceSourceAll*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:FREQ:SWEep:SRC
driver.trace.freq.sweep.src.set(freq_source = enums.TraceSourceAll.HOLD, trace␣
↪= repcap.Trace.Default)
```

Determines the source of a trace for display in frequency measurement mode.

**param freq_source**
OFF| SEN1| SEN2| SEN3| SEN4| HOLD| REF| ON ON|OFF Activates ofr deactivates the display of a trace. SEN1|SEN2|SEN3|SEN4 Activates the measurement results display of the sensor that is assigned to the trace. REF Selects a reference trace. HOLD Freezes the measurement results display of the sensor that is assigned to the trace.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

### 6.23.2 Pow

**class PowCls**

Pow 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.trace.pow.clone()
```

#### Subgroups

##### 6.23.2.1 Sweep

**class SweepCls**

Sweep 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.trace.pow.sweep.clone()
```

#### Subgroups

##### 6.23.2.1.1 Src

#### SCPI Command :

```
TRACe<CH>:POW:SWEep:SRC
```

**class SrcCls**

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

**get**(*trace=Trace.Default*) → TraceSourceAll

```
# SCPI: TRACe<CH>:POW:SWEep:SRC
value: enums.TraceSourceAll = driver.trace.pow.sweep.src.get(trace = repcap.
↪Trace.Default)
```

Determines the trace source of a trace for display in power measurement mode.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
pow_source: OFF| SEN1| SEN2| SEN3| SEN4| HOLD| REF| ON ON|OFF Activates ofr deactivates the display of a trace. SEN1|SEN2|SEN3|SEN4 Activates the measurement results display of the sensor that is assigned to the trace. REF Selects a reference

trace. HOLD Freezes the measurement results display of the sensor that is assigned to the trace.

**set**(*pow_source: TraceSourceAll*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:POW:SWEep:SRC
driver.trace.pow.sweep.src.set(pow_source = enums.TraceSourceAll.HOLD, trace =␣
↪repcap.Trace.Default)
```

Determines the trace source of a trace for display in power measurement mode.

**param pow_source**
OFF| SEN1| SEN2| SEN3| SEN4| HOLD| REF| ON ON|OFF Activates ofr deactivates the display of a trace. SEN1|SEN2|SEN3|SEN4 Activates the measurement results display of the sensor that is assigned to the trace. REF Selects a reference trace. HOLD Freezes the measurement results display of the sensor that is assigned to the trace.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

### 6.23.3 Power

**class PowerCls**

Power commands group definition. 55 total commands, 1 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.23.3.1 Sweep

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:COPY
```

**class SweepCls**

Sweep commands group definition. 55 total commands, 6 Subgroups, 1 group commands

**copy**(*copy: MeasRespTraceCopyDest*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:COPY
driver.trace.power.sweep.copy(copy = enums.MeasRespTraceCopyDest.REFerence,␣
↪trace = repcap.Trace.Default)
```

Stores the selected trace data as reference trace.

**param copy**
REFerence

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

### Cloning the Group

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

### Subgroups

#### 6.23.3.1.1 Color

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:COLor
```

**class ColorCls**

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

**get**(*trace=Trace.Default*) → MeasRespTraceColor

```
# SCPI: TRACe<CH>:[POWer]:SWEep:COLor
value: enums.MeasRespTraceColor = driver.trace.power.sweep.color.get(trace =␣
↪repcap.Trace.Default)
```

Defines the color of a trace.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
color: INVers| GRAY| YELLow| BLUE| GREen| RED| MAGenta

**set**(*color: MeasRespTraceColor*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:COLor
driver.trace.power.sweep.color.set(color = enums.MeasRespTraceColor.BLUE, trace␣
↪= repcap.Trace.Default)
```

Defines the color of a trace.

**param color**
INVers| GRAY| YELLow| BLUE| GREen| RED| MAGenta

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.2 Data

**class DataCls**

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

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.data.clone()
```

### Subgroups

#### 6.23.3.1.2.1 Points

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:DATA:POINts
```

**class PointsCls**

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

**get**(*trace=Trace.Default*) → int

```
# SCPI: TRACe<CH>:[POWer]:SWEep:DATA:POINts
value: int = driver.trace.power.sweep.data.points.get(trace = repcap.Trace.
↪Default)
```

Queries the number of measurement points of the selected trace of the current power analysis.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
points: integer Range: 10 to 1000

#### 6.23.3.1.2.2 Xvalues

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:DATA:XVALues
```

**class XvaluesCls**

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

**get**(*trace=Trace.Default*) → List[float]

```
# SCPI: TRACe<CH>:[POWer]:SWEep:DATA:XVALues
value: List[float] = driver.trace.power.sweep.data.xvalues.get(trace = repcap.
↪Trace.Default)
```

Queries the x-axis values - frequency, power or time values - of the selected trace of the current power analysis.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
xvalues: string

#### 6.23.3.1.2.3 YsValue

**SCPI Command :**

```
TRACe<CH>:[POWer]:SWEep:DATA:YSValue
```

**class YsValueCls**

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

**get**(*xvalue: float*, *trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:DATA:YSValue
value: float = driver.trace.power.sweep.data.ysValue.get(xvalue = 1.0, trace =␣
↪repcap.Trace.Default)
```

For a given x-axis value, queries the measurement (y-axis) value of the selected trace of the current power analysis.

**param xvalue**
float

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
ys_value: float

#### 6.23.3.1.2.4 Yvalues

**SCPI Command :**

```
TRACe<CH>:[POWer]:SWEep:DATA:YVALues
```

**class YvaluesCls**

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

**get**(*trace=Trace.Default*) → List[float]

```
# SCPI: TRACe<CH>:[POWer]:SWEep:DATA:YVALues
value: List[float] = driver.trace.power.sweep.data.yvalues.get(trace = repcap.
↪Trace.Default)
```

Queries the measurement (y-axis) values of the selected trace of the current power analysis.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
yvalues: string

#### 6.23.3.1.3 Feed

**SCPI Command :**

```
TRACe<CH>:[POWer]:SWEep:FEED
```

**class FeedCls**

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

**get**(*trace=Trace.Default*) → MeasRespTraceFeed

```
# SCPI: TRACe<CH>:[POWer]:SWEep:FEED
value: enums.MeasRespTraceFeed = driver.trace.power.sweep.feed.get(trace =␣
↪repcap.Trace.Default)
```

Selects the source for the trace data.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
feed: SENS1| SENS2| SENS3| REFerence| NONE| SENSor1| SENSor2| SENSor3| SENS4| SENSor4

**set**(*feed: MeasRespTraceFeed*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:FEED
driver.trace.power.sweep.feed.set(feed = enums.MeasRespTraceFeed.NONE, trace =␣
↪repcap.Trace.Default)
```

Selects the source for the trace data.

**param feed**
SENS1| SENS2| SENS3| REFerence| NONE| SENSor1| SENSor2| SENSor3| SENS4| SENSor4

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4 Measurement

**class MeasurementCls**

Measurement commands group definition. 43 total commands, 7 Subgroups, 0 group commands

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.clone()
```

##### Subgroups

###### 6.23.3.1.4.1 Fullscreen

**class FullscreenCls**

Fullscreen 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.trace.power.sweep.measurement.fullscreen.clone()
```

##### Subgroups

###### 6.23.3.1.4.2 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.fullscreen.display.clone()
```

##### Subgroups

###### 6.23.3.1.4.3 Annotation

##### SCPI Command :

```
TRACe:[POWer]:SWEep:MEASurement:FULLscreen:DISPlay:ANNotation:[STATe]
```

**class AnnotationCls**

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

**get_state**() → bool

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:FULLscreen:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.fullscreen.display.
↪annotation.get_state()
```

Selects fullscreen display of the measurement diagram on the display and in the hardcopy file.

**return**
state: 0| 1| OFF| ON

**set_state**(*state: bool*) → None

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:FULLscreen:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.fullscreen.display.annotation.set_
↪state(state = False)
```

Selects fullscreen display of the measurement diagram on the display and in the hardcopy file.

**param state**
0| 1| OFF| ON

#### 6.23.3.1.4.4 Gate

**class GateCls**

Gate 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.trace.power.sweep.measurement.gate.clone()
```

##### Subgroups

#### 6.23.3.1.4.5 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.gate.display.clone()
```

### Subgroups

#### 6.23.3.1.4.6 Annotation

### SCPI Command :

```
TRACe:[POWer]:SWEep:MEASurement:GATE:DISPlay:ANNotation:[STATe]
```

**class AnnotationCls**

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

**get_state**() → bool

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:GATE:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.gate.display.annotation.get_
↪state()
```

Activates th eindication of the time gate borders and values in the measurement diagram and in the hardcopy file. The gate settings are performed with the CALC:POW:SWE:TIME:GATE:... commands.

**return**
state: 0| 1| OFF| ON

**set_state**(*state: bool*) → None

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:GATE:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.gate.display.annotation.set_state(state =␣
↪False)
```

Activates th eindication of the time gate borders and values in the measurement diagram and in the hardcopy file. The gate settings are performed with the CALC:POW:SWE:TIME:GATE:... commands.

**param state**
0| 1| OFF| ON

#### 6.23.3.1.4.7 Marker

**class MarkerCls**

Marker 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.trace.power.sweep.measurement.marker.clone()
```

### Subgroups

#### 6.23.3.1.4.8 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.marker.display.clone()
```

### Subgroups

#### 6.23.3.1.4.9 Annotation

### SCPI Command :

```
TRACe:[POWer]:SWEep:MEASurement:MARKer:DISPlay:ANNotation:[STATe]
```

**class AnnotationCls**

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

**get_state**() → bool

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:MARKer:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.marker.display.annotation.
↪get_state()
```

Activates the indication of the markers and the marker list in the measurement diagram and in the hardcopy file.

**return**
state: 0| 1| OFF| ON

**set_state**(*state: bool*) → None

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:MARKer:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.marker.display.annotation.set_state(state␣
↪= False)
```

Activates the indication of the markers and the marker list in the measurement diagram and in the hardcopy file.

**param state**
0| 1| OFF| ON

#### 6.23.3.1.4.10 Power

**class PowerCls**

Power commands group definition. 16 total commands, 7 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.clone()
```

### Subgroups

#### 6.23.3.1.4.11 Average

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:AVERage
```

**class AverageCls**

Average commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:AVERage
value: float = driver.trace.power.sweep.measurement.power.average.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
average: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.average.clone()
```

### Subgroups

#### 6.23.3.1.4.12 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.average.display.clone()
```

### Subgroups

#### 6.23.3.1.4.13 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.average.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.14 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:AVERage:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:AVERage:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.average.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:AVERage:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.average.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.15 Hreference

**SCPI Command :**

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:HREFerence
```

**class HreferenceCls**

Hreference commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:HREFerence
value: float = driver.trace.power.sweep.measurement.power.hreference.get(trace␣
↪= repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
hreference: float Range: 0 to 100

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.hreference.clone()
```

#### Subgroups

##### 6.23.3.1.4.16 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.hreference.display.clone()
```

#### Subgroups

##### 6.23.3.1.4.17 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.hreference.display.annotation.clone()
```

#### Subgroups

##### 6.23.3.1.4.18 State

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:HREFerence:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:HREFerence:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.hreference.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:HREFerence:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.hreference.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.19 Lreference

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:LREFerence
```

**class LreferenceCls**

Lreference commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:LREFerence
value: float = driver.trace.power.sweep.measurement.power.lreference.get(trace␣
↪= repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
lreference: float Range: 0 to 100

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.lreference.clone()
```

#### Subgroups

##### 6.23.3.1.4.20 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.lreference.display.clone()
```

#### Subgroups

##### 6.23.3.1.4.21 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.lreference.display.annotation.clone()
```

#### Subgroups

##### 6.23.3.1.4.22 State

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:LREFerence:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:LREFerence:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.lreference.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:LREFerence:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.lreference.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.23 Maximum

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:MAXimum
```

**class MaximumCls**

Maximum commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:MAXimum
value: float = driver.trace.power.sweep.measurement.power.maximum.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
maximum: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.maximum.clone()
```

### Subgroups

#### 6.23.3.1.4.24 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.maximum.display.clone()
```

### Subgroups

#### 6.23.3.1.4.25 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.maximum.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.26 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:MAXimum:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:MAXimum:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.maximum.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:MAXimum:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.maximum.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.27 Minimum

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:MINimum
```

**class MinimumCls**

Minimum commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:MINimum
value: float = driver.trace.power.sweep.measurement.power.minimum.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
minimum: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.minimum.clone()
```

### Subgroups

#### 6.23.3.1.4.28 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.minimum.display.clone()
```

### Subgroups

#### 6.23.3.1.4.29 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.minimum.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.30 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:MINimum:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:MINimum:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.minimum.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:MINimum:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.minimum.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.31 Pulse

**class PulseCls**

Pulse 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.trace.power.sweep.measurement.power.pulse.clone()
```

#### Subgroups

#### 6.23.3.1.4.32 Base

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:PULSe:BASE
```

**class BaseCls**

Base commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:PULSe:BASE
value: float = driver.trace.power.sweep.measurement.power.pulse.base.get(trace␣
↪= repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
base: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.pulse.base.clone()
```

### Subgroups

#### 6.23.3.1.4.33 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.pulse.base.display.clone()
```

### Subgroups

#### 6.23.3.1.4.34 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.pulse.base.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.35 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:PULSe:BASE:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:PULSe:BASE:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.pulse.base.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:PULSe:BASE:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.pulse.base.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.36 Top

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:PULSe:TOP
```

**class TopCls**

Top commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:PULSe:TOP
value: float = driver.trace.power.sweep.measurement.power.pulse.top.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
top: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.pulse.top.clone()
```

### Subgroups

#### 6.23.3.1.4.37 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.pulse.top.display.clone()
```

### Subgroups

#### 6.23.3.1.4.38 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.pulse.top.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.39 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:PULSe:TOP:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:PULSe:TOP:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.pulse.top.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:PULSe:TOP:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.pulse.top.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.40 Reference

##### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:REFerence
```

**class ReferenceCls**

Reference commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:REFerence
value: float = driver.trace.power.sweep.measurement.power.reference.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
reference: float Range: 0 to 100

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.power.reference.clone()
```

##### Subgroups

#### 6.23.3.1.4.41 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.power.reference.display.clone()
```

### Subgroups

#### 6.23.3.1.4.42 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.power.reference.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.43 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:POWer:REFerence:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:REFerence:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.power.reference.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:POWer:REFerence:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.power.reference.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.44 Pulse

**class PulseCls**

Pulse commands group definition. 11 total commands, 7 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.pulse.clone()
```

### Subgroups

#### 6.23.3.1.4.45 All

**class AllCls**

All 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.trace.power.sweep.measurement.pulse.all.clone()
```

### Subgroups

#### 6.23.3.1.4.46 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.pulse.all.display.clone()
```

### Subgroups

#### 6.23.3.1.4.47 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.pulse.all.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.48 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:ALL:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:ALL:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.pulse.all.display.annotation.
↪state.get(trace = repcap.Trace.Default)
```

Deactivates the indication of all pulse data of the selected trace. The parameters to be indicated can be selected with the TRAC:SWE:MEAS:... commands. Only six parameters are indicated at a time. Note: This command is only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:ALL:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.pulse.all.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

Deactivates the indication of all pulse data of the selected trace. The parameters to be indicated can be selected with the TRAC:SWE:MEAS:... commands. Only six parameters are indicated at a time. Note: This command is only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.49 Dcycle

##### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:DCYCle
```

**class DcycleCls**

Dcycle commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:DCYCle
value: float = driver.trace.power.sweep.measurement.pulse.dcycle.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
dcycle: float Range: 0 to 100

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.pulse.dcycle.clone()
```

##### Subgroups

#### 6.23.3.1.4.50 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.pulse.dcycle.display.clone()
```

### Subgroups

#### 6.23.3.1.4.51 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.pulse.dcycle.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.52 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:DCYCle:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:DCYCle:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.pulse.dcycle.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:DCYCle:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.pulse.dcycle.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.53 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.pulse.display.clone()
```

### Subgroups

#### 6.23.3.1.4.54 Annotation

### SCPI Command :

```
TRACe:[POWer]:SWEep:MEASurement:PULSe:DISPlay:ANNotation:[STATe]
```

**class AnnotationCls**

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

**get_state**() → bool

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:PULSe:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.pulse.display.annotation.get_
↪state()
```

Activates the indication of the pulse data below the measurement diagram and storing the data in the hardcopy file. The parameters to be indicated can be selected with the following TRAC:SWE:MEAS:.... commands. Only six parameters are indicated at one time. Note: This command is only avalaible in time measurement mode and with R&S NRPZ81 power sensors.

**return**
state: 0| 1| OFF| ON

**set_state**(*state: bool*) → None

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:PULSe:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.pulse.display.annotation.set_state(state =␣
↪False)
```

Activates the indication of the pulse data below the measurement diagram and storing the data in the hardcopy file. The parameters to be indicated can be selected with the following TRAC:SWE:MEAS:... commands. Only six parameters are indicated at one time. Note: This command is only avalaible in time measurement mode and with R&S NRPZ81 power sensors.

**param state**
0| 1| OFF| ON

#### 6.23.3.1.4.55 Duration

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:DURation
```

**class DurationCls**

Duration commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:DURation
value: float = driver.trace.power.sweep.measurement.pulse.duration.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
duration: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.pulse.duration.clone()
```

### Subgroups

#### 6.23.3.1.4.56 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.pulse.duration.display.clone()
```

### Subgroups

#### 6.23.3.1.4.57 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.pulse.duration.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.58 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:DURation:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:DURation:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.pulse.duration.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:DURation:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.pulse.duration.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.59 Period

**SCPI Command :**

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:PERiod
```

**class PeriodCls**

Period commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:PERiod
value: float = driver.trace.power.sweep.measurement.pulse.period.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
period: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.pulse.period.clone()
```

### Subgroups

#### 6.23.3.1.4.60 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.pulse.period.display.clone()
```

### Subgroups

#### 6.23.3.1.4.61 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.pulse.period.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.62 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:PERiod:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:PERiod:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.pulse.period.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:PERiod:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.pulse.period.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.63 Separation

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:SEParation
```

**class SeparationCls**

Separation commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:SEParation
value: float = driver.trace.power.sweep.measurement.pulse.separation.get(trace␣
↪= repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
separation: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.pulse.separation.clone()
```

### Subgroups

#### 6.23.3.1.4.64 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.pulse.separation.display.clone()
```

### Subgroups

#### 6.23.3.1.4.65 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.pulse.separation.display.annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.66 State

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:SEParation:DISPlay:ANNotation:[STATe]
```

**class StateCls**

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

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:SEParation:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.pulse.separation.display.
↪annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:PULSe:SEParation:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.pulse.separation.display.annotation.state.
↪set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.67 State

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:STATe
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:PULSe:STATe
value: bool = driver.trace.power.sweep.measurement.pulse.state.get(trace =␣
↪repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: float Range: 0 to 100

#### 6.23.3.1.4.68 Standard

**class StandardCls**

Standard 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.trace.power.sweep.measurement.standard.clone()
```

### Subgroups

#### 6.23.3.1.4.69 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.standard.display.clone()
```

### Subgroups

#### 6.23.3.1.4.70 Annotation

### SCPI Command :

```
TRACe:[POWer]:SWEep:MEASurement:STANdard:DISPlay:ANNotation:[STATe]
```

**class AnnotationCls**

Annotation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:STANdard:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.standard.display.annotation.
↪get_state()
```

Selects the standard view, i.e. diagram and buttons but no lists are displayed and also stored in the hardcopy file.

**return**
state: 0| 1| OFF| ON

**set_state**(*state: bool*) → None

```
# SCPI: TRACe:[POWer]:SWEep:MEASurement:STANdard:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.standard.display.annotation.set_
↪state(state = False)
```

Selects the standard view, i.e. diagram and buttons but no lists are displayed and also stored in the hardcopy file.

**param state**
0| 1| OFF| ON

#### 6.23.3.1.4.71 Transition

**class TransitionCls**

Transition commands group definition. 12 total commands, 2 Subgroups, 0 group commands

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.clone()
```

##### Subgroups

#### 6.23.3.1.4.72 Negative

**class NegativeCls**

Negative commands group definition. 6 total commands, 3 Subgroups, 0 group commands

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.negative.clone()
```

##### Subgroups

#### 6.23.3.1.4.73 Duration

##### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:NEGative:DURation
```

**class DurationCls**

Duration commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:NEGative:DURation
value: float = driver.trace.power.sweep.measurement.transition.negative.
↪duration.get(trace = repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
duration: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.negative.duration.clone()
```

### Subgroups

#### 6.23.3.1.4.74 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.transition.negative.duration.display.
↪clone()
```

### Subgroups

#### 6.23.3.1.4.75 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.transition.negative.duration.display.
↪annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.76 State

### SCPI Command :

```
TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:DURation:DISPlay:ANNotation:[STATe]
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:DURation:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.transition.negative.duration.
↪display.annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:DURation:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.transition.negative.duration.display.
↪annotation.state.set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.77 Occurrence

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OCCurrence
```

**class OccurrenceCls**

Occurrence commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OCCurrence
value: float = driver.trace.power.sweep.measurement.transition.negative.
↪occurrence.get(trace = repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
occurrence: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.negative.occurrence.clone()
```

### Subgroups

#### 6.23.3.1.4.78 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.transition.negative.occurrence.display.
↪clone()
```

### Subgroups

#### 6.23.3.1.4.79 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.transition.negative.occurrence.display.
↪annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.80 State

### SCPI Command :

```
TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OCCurrence:DISPlay:ANNotation:[STATe]
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OCCurrence:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.transition.negative.
↪occurrence.display.annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OCCurrence:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.transition.negative.occurrence.display.
↪annotation.state.set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.81 Overshoot

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OVERshoot
```

**class OvershootCls**

Overshoot commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OVERshoot
value: float = driver.trace.power.sweep.measurement.transition.negative.
↪overshoot.get(trace = repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
overshoot: float Range: 0 to 100

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.negative.overshoot.clone()
```

#### Subgroups

#### 6.23.3.1.4.82 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.transition.negative.overshoot.display.
↪clone()
```

### Subgroups

#### 6.23.3.1.4.83 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.transition.negative.overshoot.display.
↪annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.84 State

### SCPI Command :

```
TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OVERshoot:DISPlay:ANNotation:[STATe]
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OVERshoot:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.transition.negative.
↪overshoot.display.annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:NEGative:OVERshoot:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.transition.negative.overshoot.display.
↪annotation.state.set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.85 Positive

**class PositiveCls**

Positive commands group definition. 6 total commands, 3 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.positive.clone()
```

### Subgroups

#### 6.23.3.1.4.86 Duration

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:POSitive:DURation
```

**class DurationCls**

Duration commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:POSitive:DURation
value: float = driver.trace.power.sweep.measurement.transition.positive.
↪duration.get(trace = repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
duration: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.positive.duration.clone()
```

### Subgroups

#### 6.23.3.1.4.87 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.transition.positive.duration.display.
↪clone()
```

### Subgroups

#### 6.23.3.1.4.88 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.transition.positive.duration.display.
↪annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.89 State

### SCPI Command :

```
TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:DURation:DISPlay:ANNotation:[STATe]
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:DURation:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.transition.positive.duration.
↪display.annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:DURation:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.transition.positive.duration.display.
↪annotation.state.set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.90 Occurrence

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OCCurrence
```

**class OccurrenceCls**

Occurrence commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OCCurrence
value: float = driver.trace.power.sweep.measurement.transition.positive.
↪occurrence.get(trace = repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
occurrence: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.positive.occurrence.clone()
```

### Subgroups

#### 6.23.3.1.4.91 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.transition.positive.occurrence.display.
↪clone()
```

### Subgroups

#### 6.23.3.1.4.92 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.transition.positive.occurrence.display.
↪annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.93 State

### SCPI Command :

```
TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OCCurrence:DISPlay:ANNotation:[STATe]
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OCCurrence:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.transition.positive.
↪occurrence.display.annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OCCurrence:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.transition.positive.occurrence.display.
↪annotation.state.set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.4.94 Overshoot

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OVERshoot
```

**class OvershootCls**

Overshoot commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OVERshoot
value: float = driver.trace.power.sweep.measurement.transition.positive.
↪overshoot.get(trace = repcap.Trace.Default)
```

The above listed commands query the measured pulse parameter values. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
overshoot: float Range: 0 to 100

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.measurement.transition.positive.overshoot.clone()
```

### Subgroups

#### 6.23.3.1.4.95 Display

**class DisplayCls**

Display 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.trace.power.sweep.measurement.transition.positive.overshoot.display.
↪clone()
```

### Subgroups

#### 6.23.3.1.4.96 Annotation

**class AnnotationCls**

Annotation 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.trace.power.sweep.measurement.transition.positive.overshoot.display.
↪annotation.clone()
```

### Subgroups

#### 6.23.3.1.4.97 State

### SCPI Command :

```
TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OVERshoot:DISPlay:ANNotation:[STATe]
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → bool

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OVERshoot:DISPlay:ANNotation:[STATe]
value: bool = driver.trace.power.sweep.measurement.transition.positive.
↪overshoot.display.annotation.state.get(trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: 0| 1| OFF| ON

**set**(*state: bool*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>
↪:[POWer]:SWEep:MEASurement:TRANsition:POSitive:OVERshoot:DISPlay:ANNotation:[STATe]
driver.trace.power.sweep.measurement.transition.positive.overshoot.display.
↪annotation.state.set(state = False, trace = repcap.Trace.Default)
```

The above listed commands select the pulse parameters which are indicated in the display and hardcopy file. Only six parameters can be indicated at a time. Note: These commands are only available in time measurement mode and with R&S NRP-Z81 power sensors.

**param state**
0| 1| OFF| ON

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.5 Pulse

**class PulseCls**

Pulse 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.trace.power.sweep.pulse.clone()
```

##### Subgroups

#### 6.23.3.1.5.1 Threshold

**class ThresholdCls**

Threshold 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.trace.power.sweep.pulse.threshold.clone()
```

##### Subgroups

#### 6.23.3.1.5.2 Base

##### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:BASE
```

**class BaseCls**

Base commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → MeasRespPulsThrBase

```
# SCPI: TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:BASE
value: enums.MeasRespPulsThrBase = driver.trace.power.sweep.pulse.threshold.
↪base.get(trace = repcap.Trace.Default)
```

Queries how the threshold parameters are calculated. Note: This parameter is only avalaible in time measurement mode and R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
base: VOLTage| POWer

#### 6.23.3.1.5.3 Power

**class PowerCls**

Power commands group definition. 3 total commands, 3 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.power.sweep.pulse.threshold.power.clone()
```

### Subgroups

#### 6.23.3.1.5.4 Hreference

### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:HREFerence
```

**class HreferenceCls**

Hreference commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:HREFerence
value: float = driver.trace.power.sweep.pulse.threshold.power.hreference.
↪get(trace = repcap.Trace.Default)
```

Queries the upper threshold level of the overall pulse level. The distal power defines the end of the rising edge and the start of the falling edge of the pulse. Note: This parameter is only avalaible in time measurement mode and R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
hreference: float Range: 0.0 to 100.0

**set**(*hreference: float*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:HREFerence
driver.trace.power.sweep.pulse.threshold.power.hreference.set(hreference = 1.0,␣
↪trace = repcap.Trace.Default)
```

Queries the upper threshold level of the overall pulse level. The distal power defines the end of the rising edge and the start of the falling edge of the pulse. Note: This parameter is only avalaible in time measurement mode and R&S NRP-Z81 power sensors.

**param hreference**
float Range: 0.0 to 100.0

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.5.5 Lreference

#### SCPI Command :

```
TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:LREFerence
```

**class LreferenceCls**

Lreference commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:LREFerence
value: float = driver.trace.power.sweep.pulse.threshold.power.lreference.
↪get(trace = repcap.Trace.Default)
```

Queries the lower medial threshold level of the overall pulse level. The proximal power defines the start of the rising edge and the end of the falling edge of the pulse. Note: This parameter is only avalaible in time measurement mode and R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
lreference: float Range: 0.0 to 100.0

**set**(*lreference: float*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:LREFerence
driver.trace.power.sweep.pulse.threshold.power.lreference.set(lreference = 1.0,␣
↪trace = repcap.Trace.Default)
```

Queries the lower medial threshold level of the overall pulse level. The proximal power defines the start of the rising edge and the end of the falling edge of the pulse. Note: This parameter is only avalaible in time measurement mode and R&S NRP-Z81 power sensors.

**param lreference**
float Range: 0.0 to 100.0

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.5.6 Reference

**SCPI Command :**

```
TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:REFerence
```

**class ReferenceCls**

Reference commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → float

```
# SCPI: TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:REFerence
value: float = driver.trace.power.sweep.pulse.threshold.power.reference.
↪get(trace = repcap.Trace.Default)
```

Queries the medial threshold level of the overall pulse level. This level is used to define the pulse width and pulse period. Note: This parameter is only avalaible in time measurement mode and R&S NRP-Z81 power sensors.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
reference: float Range: 0.0 to 100.0

**set**(*reference: float*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:PULSe:THReshold:POWer:REFerence
driver.trace.power.sweep.pulse.threshold.power.reference.set(reference = 1.0,␣
↪trace = repcap.Trace.Default)
```

Queries the medial threshold level of the overall pulse level. This level is used to define the pulse width and pulse period. Note: This parameter is only avalaible in time measurement mode and R&S NRP-Z81 power sensors.

**param reference**
float Range: 0.0 to 100.0

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

#### 6.23.3.1.6 State

**SCPI Command :**

```
TRACe<CH>:[POWer]:SWEep:STATe
```

**class StateCls**

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → MeasRespTraceState

```
# SCPI: TRACe<CH>:[POWer]:SWEep:STATe
value: enums.MeasRespTraceState = driver.trace.power.sweep.state.get(trace =␣
↪repcap.Trace.Default)
```

Activates the selected trace.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
state: OFF| ON| HOLD

**set**(*state: MeasRespTraceState*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:[POWer]:SWEep:STATe
driver.trace.power.sweep.state.set(state = enums.MeasRespTraceState.HOLD, trace␣
↪= repcap.Trace.Default)
```

Activates the selected trace.

**param state**
OFF| ON| HOLD

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

### 6.23.4 Time

**class TimeCls**

Time 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.trace.time.clone()
```

#### Subgroups

##### 6.23.4.1 Sweep

**class SweepCls**

Sweep 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.trace.time.sweep.clone()
```

### Subgroups

#### 6.23.4.1.1 Src

### SCPI Command :

```
TRACe<CH>:TIME:SWEep:SRC
```

**class SrcCls**

Src commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*trace=Trace.Default*) → TraceSourceAll

```
# SCPI: TRACe<CH>:TIME:SWEep:SRC
value: enums.TraceSourceAll = driver.trace.time.sweep.src.get(trace = repcap.
↪Trace.Default)
```

Determines the trace source of a trace for display in time measurement mode.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

**return**
time_source: OFF| SEN1| SEN2| SEN3| SEN4| HOLD| REF| ON ON|OFF Activates ofr deactivates the display of a trace. SEN1|SEN2|SEN3|SEN4 Activates the measurement results display of the sensor that is assigned to the trace. REF Selects a reference trace. HOLD Freezes the measurement results display of the sensor that is assigned to the trace.

**set**(*time_source: TraceSourceAll*, *trace=Trace.Default*) → None

```
# SCPI: TRACe<CH>:TIME:SWEep:SRC
driver.trace.time.sweep.src.set(time_source = enums.TraceSourceAll.HOLD, trace␣
↪= repcap.Trace.Default)
```

Determines the trace source of a trace for display in time measurement mode.

**param time_source**
OFF| SEN1| SEN2| SEN3| SEN4| HOLD| REF| ON ON|OFF Activates ofr deactivates the display of a trace. SEN1|SEN2|SEN3|SEN4 Activates the measurement results display of the sensor that is assigned to the trace. REF Selects a reference trace. HOLD Freezes the measurement results display of the sensor that is assigned to the trace.

**param trace**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trace')

## 6.24 Trigger<InputIx>

### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.trigger.repcap_inputIx_get()
driver.trigger.repcap_inputIx_set(repcap.InputIx.Nr1)
```

**class TriggerCls**

Trigger commands group definition. 14 total commands, 6 Subgroups, 0 group commands Repeated Capability: InputIx, default value after init: InputIx.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.clone()
```

### Subgroups

### 6.24.1 FpSweep

**class FpSweepCls**

FpSweep 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.trigger.fpSweep.clone()
```

### Subgroups

#### 6.24.1.1 Source

### SCPI Command :

```
TRIGger<HW>:FPSWeep:SOURce
```

**class SourceCls**

Source commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → SingExtAuto

```
# SCPI: TRIGger<HW>:FPSWeep:SOURce
value: enums.SingExtAuto = driver.trigger.fpSweep.source.get(inputIx = repcap.
↪InputIx.Default)
```

Selects the trigger source for the combined RF frequency / level sweep. The parameter names correspond to the manual control. If needed, see table Table 'Cross-reference between the manual and remote control' for selecting the trigger source with SCPI compliant parameter names.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
fp_trig_source: AUTO| IMMediate | SINGle| BUS| EXTernal | EAUTo AUTO|IMMediate Executes the combined RF sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. as soon as a sweep is completed, the next one starts immediately. SINGle|BUS Executes one complete sweep cycle triggered by the GPIB commands [:SOURcehw]:SWEep:COMBined:EXECute or *TRG. The mode has to be set to [:SOURcehw]:SWEep:COMBined:MODE AUTO. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. As soon as one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency and level value pairs, a third trigger event starts the trigger at the start values, and so on.

**set**(*fp_trig_source: SingExtAuto*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:FPSWeep:SOURce
driver.trigger.fpSweep.source.set(fp_trig_source = enums.SingExtAuto.AUTO,␣
↪inputIx = repcap.InputIx.Default)
```

Selects the trigger source for the combined RF frequency / level sweep. The parameter names correspond to the manual control. If needed, see table Table 'Cross-reference between the manual and remote control' for selecting the trigger source with SCPI compliant parameter names.

**param fp_trig_source**
AUTO| IMMediate | SINGle| BUS| EXTernal | EAUTo AUTO|IMMediate Executes the combined RF sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. as soon as a sweep is completed, the next one starts immediately. SINGle|BUS Executes one complete sweep cycle triggered by the GPIB commands [:SOURcehw]:SWEep:COMBined:EXECute or *TRG. The mode has to be set to [:SOURcehw]:SWEep:COMBined:MODE AUTO. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. As soon as one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency and level value pairs, a third trigger event starts the trigger at the start values, and so on.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### 6.24.2 FreqSweep

**class FreqSweepCls**

FreqSweep commands group definition. 3 total commands, 2 Subgroups, 0 group commands

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.freqSweep.clone()
```

#### Subgroups

##### 6.24.2.1 Immediate

#### SCPI Command :

```
TRIGger<HW>:FSWeep:[IMMediate]
```

**class ImmediateCls**

Immediate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**set**(*inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:FSWeep:[IMMediate]
driver.trigger.freqSweep.immediate.set(inputIx = repcap.InputIx.Default)


    INTRO_CMD_HELP: Performs a single sweep and immediately starts the␣
↪activated, corresponding sweep:

    - FSWeep - RF frequency
    - PSWeep - RF level
    - LFFSweep - LF frequency
    - SWEep - all sweeps
    INTRO_CMD_HELP: Effective in the following configuration:

    - TRIG:FSW|LFFS|PSW|[:SWE]:SOUR SING
    - SOUR:SWE:FREQ|POW:MODE AUTO or SOUR:LFO:SWE:[FREQ:]MODE AUTO
```

Alternatively, you can use the IMMediate command instead of the respective SWEep:[FREQ:]|POW:EXECute command.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**set_with_opc**(*inputIx=InputIx.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.24.2.2 Source

**SCPI Command :**

```
TRIGger<HW>:FSWeep:SOURce
```

**class SourceCls**

Source commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → SingExtAuto

```
# SCPI: TRIGger<HW>:FSWeep:SOURce
value: enums.SingExtAuto = driver.trigger.freqSweep.source.get(inputIx = repcap.
↪InputIx.Default)


    INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

    - FSWeep - RF frequency
    - LFFSweep - LF frequency
    - PSWeep - RF level
    - SWEep - all sweeps
```

The source names of the parameters correspond to the values provided in manual control of the instrument. They differ from the SCPI-compliant names, but the instrument accepts both variants. Use the SCPI name, if compatibility is an important issue. Find the corresponding SCPI-compliant commands in Cross-reference between the manual and remote control.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
source: AUTO| IMMediate | SINGle| BUS | EXTernal | EAUTo AUTO [IMMediate] Executes a sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. when a sweep is completed, the next one starts immediately. SINGle [BUS] Executes one complete sweep cycle. The following commands initiate a trigger event: *TRG [:SOURcehw]:SWEep:POWer:EXECute [:SOURcehw]:SWEep[:FREQuency]:EXECute method RsSmab.Trigger.Sweep.Immediate.set, method RsSmab.Trigger.Psweep.Immediate.set and method RsSmab.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]:SWEep[:FREQuency]:MODEAUTO|STEP [:SOURcehw]:LFOutput:SWEep[:FREQuency]:MODEAUTO|STEP In step mode (STEP) , the instrument executes only one step. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. When one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency, a third trigger event starts the trigger at the start frequency, and so on.

**set**(*source: SingExtAuto*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:FSWeep:SOURce
driver.trigger.freqSweep.source.set(source = enums.SingExtAuto.AUTO, inputIx =␣
↪repcap.InputIx.Default)
```

(continues on next page)

(continued from previous page)

```
INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

- FSWeep - RF frequency
- LFFSweep - LF frequency
- PSWeep - RF level
- SWEep - all sweeps
```

The source names of the parameters correspond to the values provided in manual control of the instrument. They differ from the SCPI-compliant names, but the instrument accepts both variants. Use the SCPI name, if compatibility is an important issue. Find the corresponding SCPI-compliant commands in Cross-reference between the manual and remote control.

**param source**
AUTO| IMMediate | SINGle| BUS | EXTernal | EAUTo AUTO [IMMediate] Executes a sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. when a sweep is completed, the next one starts immediately. SINGle [BUS] Executes one complete sweep cycle. The following commands initiate a trigger event: *TRG [:SOURcehw]:SWEep:POWer:EXECute [:SOURcehw]:SWEep[:FREQuency]:EXECute method RsSmab.Trigger.Sweep.Immediate.set, method RsSmab.Trigger.Psweep.Immediate.set and method RsSmab.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]:SWEep[:FREQuency]:MODEAUTO|STEP [:SOURcehw]:LFOutput:SWEep[:FREQuency]:MODEAUTO|STEP In step mode (STEP) , the instrument executes only one step. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. When one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency, a third trigger event starts the trigger at the start frequency, and so on.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.freqSweep.source.clone()
```

### Subgroups

#### 6.24.2.2.1 Advanced

### SCPI Command :

```
TRIGger<HW>:FSWeep:SOURce:ADVanced
```

**class AdvancedCls**

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → TrigSweepImmBusExt

```
# SCPI: TRIGger<HW>:FSWeep:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.trigger.freqSweep.source.advanced.
↪get(inputIx = repcap.InputIx.Default)
```

No command help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
fs_trig_source_adv: No help available

**set**(*fs_trig_source_adv: TrigSweepImmBusExt*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:FSWeep:SOURce:ADVanced
driver.trigger.freqSweep.source.advanced.set(fs_trig_source_adv = enums.
↪TrigSweepImmBusExt.BUS, inputIx = repcap.InputIx.Default)
```

No command help available

**param fs_trig_source_adv**
No help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### 6.24.3 LffSweep

#### SCPI Command :

```
TRIGger<HW>:LFFSweep
```

**class LffSweepCls**

LffSweep commands group definition. 4 total commands, 2 Subgroups, 1 group commands

**set**(*inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:LFFSweep
driver.trigger.lffSweep.set(inputIx = repcap.InputIx.Default)


    INTRO_CMD_HELP: Executes an LF frequency sweep in the following␣
↪configuration:

    - method RsSmab.Trigger.LffSweep.Source.set SING
    - LFO:SWE:MODE AUTO

    :param inputIx: optional repeated capability selector. Default value: Nr1␣
↪(settable in the interface 'Trigger')
```

**set_with_opc**(*inputIx=InputIx.Default*, *opc_timeout_ms: int = -1*) → None

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lffSweep.clone()
```

### Subgroups

#### 6.24.3.1 Immediate

### SCPI Command :

```
TRIGger<HW>:LFFSweep:IMMediate
```

**class ImmediateCls**

Immediate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**set**(*inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:LFFSweep:IMMediate
driver.trigger.lffSweep.immediate.set(inputIx = repcap.InputIx.Default)


    INTRO_CMD_HELP: Performs a single sweep and immediately starts the␣
↪activated, corresponding sweep:

    - FSWeep - RF frequency
    - PSWeep - RF level
    - LFFSweep - LF frequency
    - SWEep - all sweeps
    INTRO_CMD_HELP: Effective in the following configuration:

    - TRIG:FSW|LFFS|PSW|[:SWE]:SOUR SING
    - SOUR:SWE:FREQ|POW:MODE AUTO or SOUR:LFO:SWE:[FREQ:]MODE AUTO
```

Alternatively, you can use the IMMediate command instead of the respective SWEep:[FREQ:]|POW:EXECute command.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**set_with_opc**(*inputIx=InputIx.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.24.3.2 Source

**SCPI Command :**

```
TRIGger<HW>:LFFSweep:SOURce
```

**class SourceCls**

Source commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → SingExtAuto

```
# SCPI: TRIGger<HW>:LFFSweep:SOURce
value: enums.SingExtAuto = driver.trigger.lffSweep.source.get(inputIx = repcap.
↪InputIx.Default)


    INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

    - FSWeep - RF frequency
    - LFFSweep - LF frequency
    - PSWeep - RF level
    - SWEep - all sweeps
```

The source names of the parameters correspond to the values provided in manual control of the instrument. They differ from the SCPI-compliant names, but the instrument accepts both variants. Use the SCPI name, if compatibility is an important issue. Find the corresponding SCPI-compliant commands in Cross-reference between the manual and remote control.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
source: AUTO| IMMediate | SINGle| BUS | EXTernal | EAUTo AUTO [IMMediate] Executes a sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. when a sweep is completed, the next one starts immediately. SINGle [BUS] Executes one complete sweep cycle. The following commands initiate a trigger event: *TRG [:SOURcehw]:SWEep:POWer:EXECute [:SOURcehw]:SWEep[:FREQuency]:EXECute method RsSmab.Trigger.Sweep.Immediate.set, method RsSmab.Trigger.Psweep.Immediate.set and method RsSmab.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]:SWEep[:FREQuency]:MODEAUTO|STEP [:SOURcehw]:LFOutput:SWEep[:FREQuency]:MODEAUTO|STEP In step mode (STEP) , the instrument executes only one step. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. When one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency, a third trigger event starts the trigger at the start frequency, and so on.

**set**(*source: SingExtAuto*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:LFFSweep:SOURce
driver.trigger.lffSweep.source.set(source = enums.SingExtAuto.AUTO, inputIx =␣
↪repcap.InputIx.Default)
```

(continues on next page)

(continued from previous page)

```
INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

- FSWeep - RF frequency
- LFFSweep - LF frequency
- PSWeep - RF level
- SWEep - all sweeps
```

The source names of the parameters correspond to the values provided in manual control of the instrument. They differ from the SCPI-compliant names, but the instrument accepts both variants. Use the SCPI name, if compatibility is an important issue. Find the corresponding SCPI-compliant commands in Cross-reference between the manual and remote control.

**param source**
AUTO| IMMediate | SINGle| BUS | EXTernal | EAUTo AUTO [IMMediate] Executes a sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. when a sweep is completed, the next one starts immediately. SINGle [BUS] Executes one complete sweep cycle. The following commands initiate a trigger event: *TRG [:SOURcehw]:SWEep:POWer:EXECute [:SOURcehw]:SWEep[:FREQuency]:EXECute method RsSmab.Trigger.Sweep.Immediate.set, method RsSmab.Trigger.Psweep.Immediate.set and method RsSmab.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]:SWEep[:FREQuency]:MODEAUTO|STEP [:SOURcehw]:LFOutput:SWEep[:FREQuency]:MODEAUTO|STEP In step mode (STEP) , the instrument executes only one step. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. When one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency, a third trigger event starts the trigger at the start frequency, and so on.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lffSweep.source.clone()
```

### Subgroups

#### 6.24.3.2.1 Advanced

### SCPI Command :

```
TRIGger<HW>:LFFSweep:SOURce:ADVanced
```

**class AdvancedCls**

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → TrigSweepImmBusExt

```
# SCPI: TRIGger<HW>:LFFSweep:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.trigger.lffSweep.source.advanced.
↪get(inputIx = repcap.InputIx.Default)
```

No command help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
lf_trig_sour_adv: No help available

**set**(*lf_trig_sour_adv: TrigSweepImmBusExt*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:LFFSweep:SOURce:ADVanced
driver.trigger.lffSweep.source.advanced.set(lf_trig_sour_adv = enums.
↪TrigSweepImmBusExt.BUS, inputIx = repcap.InputIx.Default)
```

No command help available

**param lf_trig_sour_adv**
No help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### 6.24.4 ListPy

**class ListPyCls**

ListPy 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.trigger.listPy.clone()
```

#### Subgroups

##### 6.24.4.1 Source

**class SourceCls**

Source 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.trigger.listPy.source.clone()
```

### Subgroups

#### 6.24.4.1.1 Advanced

### SCPI Command :

```
TRIGger<HW>:LIST:SOURce:ADVanced
```

**class AdvancedCls**

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → TrigSweepImmBusExt

```
# SCPI: TRIGger<HW>:LIST:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.trigger.listPy.source.advanced.
↪get(inputIx = repcap.InputIx.Default)
```

No command help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
start_trig_adv: No help available

**set**(*start_trig_adv: TrigSweepImmBusExt*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:LIST:SOURce:ADVanced
driver.trigger.listPy.source.advanced.set(start_trig_adv = enums.
↪TrigSweepImmBusExt.BUS, inputIx = repcap.InputIx.Default)
```

No command help available

**param start_trig_adv**
No help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### 6.24.5 Psweep

**class PsweepCls**

Psweep commands group definition. 3 total commands, 2 Subgroups, 0 group commands

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.psweep.clone()
```

#### Subgroups

##### 6.24.5.1 Immediate

#### SCPI Command :

```
TRIGger<HW>:PSWeep:[IMMediate]
```

**class ImmediateCls**

Immediate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**set**(*inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:PSWeep:[IMMediate]
driver.trigger.psweep.immediate.set(inputIx = repcap.InputIx.Default)


    INTRO_CMD_HELP: Performs a single sweep and immediately starts the␣
↪activated, corresponding sweep:

    - FSWeep - RF frequency
    - PSWeep - RF level
    - LFFSweep - LF frequency
    - SWEep - all sweeps
    INTRO_CMD_HELP: Effective in the following configuration:

    - TRIG:FSW|LFFS|PSW|[:SWE]:SOUR SING
    - SOUR:SWE:FREQ|POW:MODE AUTO or SOUR:LFO:SWE:[FREQ:]MODE AUTO
```

Alternatively, you can use the IMMediate command instead of the respective SWEep:[FREQ:]|POW:EXECute command.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**set_with_opc**(*inputIx=InputIx.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.24.5.2 Source

**SCPI Command :**

```
TRIGger<HW>:PSWeep:SOURce
```

**class SourceCls**

Source commands group definition. 2 total commands, 1 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → SingExtAuto

```
# SCPI: TRIGger<HW>:PSWeep:SOURce
value: enums.SingExtAuto = driver.trigger.psweep.source.get(inputIx = repcap.
↪InputIx.Default)


    INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

    - FSWeep - RF frequency
    - LFFSweep - LF frequency
    - PSWeep - RF level
    - SWEep - all sweeps
```

The source names of the parameters correspond to the values provided in manual control of the instrument. They differ from the SCPI-compliant names, but the instrument accepts both variants. Use the SCPI name, if compatibility is an important issue. Find the corresponding SCPI-compliant commands in Cross-reference between the manual and remote control.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
source: AUTO| IMMediate | SINGle| BUS | EXTernal | EAUTo AUTO [IMMediate] Executes a sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. when a sweep is completed, the next one starts immediately. SINGle [BUS] Executes one complete sweep cycle. The following commands initiate a trigger event: *TRG [:SOURcehw]:SWEep:POWer:EXECute [:SOURcehw]:SWEep[:FREQuency]:EXECute method RsSmab.Trigger.Sweep.Immediate.set, method RsSmab.Trigger.Psweep.Immediate.set and method RsSmab.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]:SWEep[:FREQuency]:MODEAUTO|STEP [:SOURcehw]:LFOutput:SWEep[:FREQuency]:MODEAUTO|STEP In step mode (STEP) , the instrument executes only one step. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. When one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency, a third trigger event starts the trigger at the start frequency, and so on.

**set**(*source: SingExtAuto*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:PSWeep:SOURce
driver.trigger.psweep.source.set(source = enums.SingExtAuto.AUTO, inputIx =␣
↪repcap.InputIx.Default)
```

(continues on next page)

(continued from previous page)

```
INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

- FSWeep - RF frequency
- LFFSweep - LF frequency
- PSWeep - RF level
- SWEep - all sweeps
```

The source names of the parameters correspond to the values provided in manual control of the instrument. They differ from the SCPI-compliant names, but the instrument accepts both variants. Use the SCPI name, if compatibility is an important issue. Find the corresponding SCPI-compliant commands in Cross-reference between the manual and remote control.

**param source**
AUTO| IMMediate | SINGle| BUS | EXTernal | EAUTo AUTO [IMMediate] Executes a sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. when a sweep is completed, the next one starts immediately. SINGle [BUS] Executes one complete sweep cycle. The following commands initiate a trigger event: *TRG [:SOURcehw]:SWEep:POWer:EXECute [:SOURcehw]:SWEep[:FREQuency]:EXECute method RsSmab.Trigger.Sweep.Immediate.set, method RsSmab.Trigger.Psweep.Immediate.set and method RsSmab.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]:SWEep[:FREQuency]:MODEAUTO|STEP [:SOURcehw]:LFOutput:SWEep[:FREQuency]:MODEAUTO|STEP In step mode (STEP) , the instrument executes only one step. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. When one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency, a third trigger event starts the trigger at the start frequency, and so on.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.psweep.source.clone()
```

### Subgroups

#### 6.24.5.2.1 Advanced

### SCPI Command :

```
TRIGger<HW>:PSWeep:SOURce:ADVanced
```

**class AdvancedCls**

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**get**(*inputIx=InputIx.Default*) → TrigSweepImmBusExt

```
# SCPI: TRIGger<HW>:PSWeep:SOURce:ADVanced
value: enums.TrigSweepImmBusExt = driver.trigger.psweep.source.advanced.
↪get(inputIx = repcap.InputIx.Default)
```

No command help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**return**
ps_trig_source_adv: No help available

**set**(*ps_trig_source_adv: TrigSweepImmBusExt*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:PSWeep:SOURce:ADVanced
driver.trigger.psweep.source.advanced.set(ps_trig_source_adv = enums.
↪TrigSweepImmBusExt.BUS, inputIx = repcap.InputIx.Default)
```

No command help available

**param ps_trig_source_adv**
No help available

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

### 6.24.6 Sweep

**class SweepCls**

Sweep 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.trigger.sweep.clone()
```

#### Subgroups

#### 6.24.6.1 Immediate

#### SCPI Command :

```
TRIGger<HW>:[SWEep]:[IMMediate]
```

**class ImmediateCls**

Immediate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**set**(*inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:[SWEep]:[IMMediate]
driver.trigger.sweep.immediate.set(inputIx = repcap.InputIx.Default)


    INTRO_CMD_HELP: Performs a single sweep and immediately starts the␣
↪activated, corresponding sweep:

    - FSWeep - RF frequency
    - PSWeep - RF level
    - LFFSweep - LF frequency
    - SWEep - all sweeps
    INTRO_CMD_HELP: Effective in the following configuration:

    - TRIG:FSW|LFFS|PSW|[:SWE]:SOUR SING
    - SOUR:SWE:FREQ|POW:MODE AUTO or SOUR:LFO:SWE:[FREQ:]MODE AUTO
```

Alternatively, you can use the IMMediate command instead of the respective SWEep:[FREQ:]|POW:EXECute command.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

**set_with_opc**(*inputIx=InputIx.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.24.6.2 Source

**SCPI Command :**

```
TRIGger<HW>:[SWEep]:SOURce
```

**class SourceCls**

Source commands group definition. 1 total commands, 0 Subgroups, 1 group commands

**set**(*source: SingExtAuto*, *inputIx=InputIx.Default*) → None

```
# SCPI: TRIGger<HW>:[SWEep]:SOURce
driver.trigger.sweep.source.set(source = enums.SingExtAuto.AUTO, inputIx =␣
↪repcap.InputIx.Default)


    INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

    - FSWeep - RF frequency
    - LFFSweep - LF frequency
    - PSWeep - RF level
    - SWEep - all sweeps
```

The source names of the parameters correspond to the values provided in manual control of the instrument. They differ from the SCPI-compliant names, but the instrument accepts both variants. Use the SCPI name, if compatibility is an important issue. Find the corresponding SCPI-compliant commands in Cross-reference between the manual and remote control.

**param source**
AUTO| IMMediate | SINGle| BUS | EXTernal | EAUTo AUTO [IMMediate] Executes a sweep automatically. In this free-running mode, the trigger condition is met continuously. I.e. when a sweep is completed, the next one starts immediately. SINGle [BUS] Executes one complete sweep cycle. The following commands initiate a trigger event: *TRG [:SOURcehw]:SWEep:POWer:EXECute [:SOURcehw]:SWEep[:FREQuency]:EXECute method RsSmab.Trigger.Sweep.Immediate.set, method RsSmab.Trigger.Psweep.Immediate.set and method RsSmab.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]:SWEep[:FREQuency]:MODEAUTO|STEP [:SOURcehw]:LFOutput:SWEep[:FREQuency]:MODEAUTO|STEP In step mode (STEP) , the instrument executes only one step. EXTernal An external signal triggers the sweep. EAUTo An external signal triggers the sweep. When one sweep is finished, the next sweep starts. A second trigger event stops the sweep at the current frequency, a third trigger event starts the trigger at the start frequency, and so on.

**param inputIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trigger')

## 6.25 Unit

### SCPI Commands :

```
UNIT:ANGLe
UNIT:POWer
UNIT:VELocity
```

**class UnitCls**

Unit commands group definition. 3 total commands, 0 Subgroups, 3 group commands

**get_angle**() → UnitAngle

```
# SCPI: UNIT:ANGLe
value: enums.UnitAngle = driver.unit.get_angle()
```

Sets the default unit for phase modulation angle. The command affects no other parameters, such as RF phase, or the manual control or display.

**return**
angle: DEGree| DEGRee| RADian

**get_power**() → UnitPower

```
# SCPI: UNIT:POWer
value: enums.UnitPower = driver.unit.get_power()
```

Sets the default unit for all power parameters. This setting affects the GUI, as well as all remote control commands that determine power values.

**return**
power: V| DBUV| DBM

**get_velocity**() → UnitSpeed

```
# SCPI: UNIT:VELocity
value: enums.UnitSpeed = driver.unit.get_velocity()
```

No command help available

**return**
velocity: No help available

**set_angle**(*angle: UnitAngle*) → None

```
# SCPI: UNIT:ANGLe
driver.unit.set_angle(angle = enums.UnitAngle.DEGree)
```

Sets the default unit for phase modulation angle. The command affects no other parameters, such as RF phase, or the manual control or display.

**param angle**
DEGree| DEGRee| RADian

**set_power**(*power: UnitPower*) → None

```
# SCPI: UNIT:POWer
driver.unit.set_power(power = enums.UnitPower.DBM)
```

Sets the default unit for all power parameters. This setting affects the GUI, as well as all remote control commands that determine power values.

**param power**
V| DBUV| DBM

**set_velocity**(*velocity: UnitSpeed*) → None

```
# SCPI: UNIT:VELocity
driver.unit.set_velocity(velocity = enums.UnitSpeed.KMH)
```

No command help available

**param velocity**
No help available

CHAPTER

# SEVEN

# RSSMAB UTILITIES

**class Utilities**

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsSmab.utilities`

**property logger:** *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsSmab.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 RsSmab instance with new VISA session, the session gets a new thread lock. You can assign it to other RsSmab sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsSmab 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

# EIGHT

# RSSMAB 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

# NINE

# RSSMAB 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

TEN

# INDEX

# INDEX

## Symbols

## A



## B



## C

## D



## E



## F



## G



## H

## I



## K



## L



## M



## O



## R



## S

## T

# U