RsSmcv

Release 5.20.43.18

Rohde & Schwarz

Mar 25, 2024

CONTENTS:


SMCV

CHAPTER

ONE

REVISION HISTORY

1.1 RsSmcv

Rohde & Schwarz SMCV100B Vector Signal Generator RsSmcv instrument driver.

Basic Hello-World code:

```
from RsSmcv import *

instr = RsSmcv('TCPIP::192.168.56.101::hislip0', reset=True)
idn = instr.query_str('*IDN?')
print('Hello, I am: ' + idn)
```

Supported instruments: SMCV100B

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

Documentation: https://RsSmcv.readthedocs.io/

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

1.1.1 Version history

Latest release notes summary: Update for FW 5.20.043

Version 5.20.43

- Update for FW 5.20.043

Version 5.0.124.16

- Updated core to the newest template.
- Added DigitalModulation Interface.

Version 5.0.122.13

- Update for FW 5.00.122

Version 4.80.2.11

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

Version 4.80.2.6

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsSmcv 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 RsSmcv example:

```
"""Getting started - how to work with RsSmcv Python package.
This example performs basic RF settings on an SMCV100B instrument.
It shows the RsSmcv calls and their corresponding SCPI commands.
Notice that the python RsSmcv interfaces track the SCPI commands syntax."""

from RsSmcv import *

# Open the session
RsSmcv.assert_minimum_version('5.0.122')
smcv = RsSmcv('TCPIP::10.102.52.52::HISLIP')
# Greetings, stranger...
print(f'Hello, I am: {smcv.utilities.idn_string}')

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

#    SOURce:FREQuency:MODE CW
```

(continues on next page)

(continued from previous page)

```
smcv.source.frequency.set_mode(enums.FreqMode.CW)

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

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

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


# Close the session
smcv.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

RsSmcv 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 RsSmcv`

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 `RsSmcv` 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 RsSmcv offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsSmcv needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsSmcv package to your computer from the pypi.org: https://pypi.org/project/RsSmcv/#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\RsSmcv-5.20.43.18.tar`

2.3 Finding Available Instruments

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

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

from RsSmcv import *

# Use the instr_list string items as resource names in the RsSmcv constructor
instr_list = RsSmcv.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 RsSmcv 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 = RsSmcv.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

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

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

- Installed RsSmcv Python module Version 5.20.43 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsSmcv import *

# A good practice is to assure that you have a certain minimum version installed
RsSmcv.assert_minimum_version('5.20.43')
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 = RsSmcv(resource_string_1)

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

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

from RsSmcv import *

# Force use of the Rs Visa. For NI Visa, use the "SelectVisa='ni'"
driver = RsSmcv('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, RsSmcv has something for you too - **no Visa installation raw LAN socket**:

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

from RsSmcv import *

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

More `option_string` tokens are separated by comma:

```
driver = RsSmcv('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 RsSmcv objects:

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

from RsSmcv import *

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

driver = RsSmcv('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 RsSmcv import *

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

driver = RsSmcv('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

RsSmcv 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 RsSmcv 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 RsSmcv import *

driver = None
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsSmcv('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 RsSmcv exceptions
    print(e.args[0])
    print('Some other RsSmcv 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 RsSmcv, 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 RsSmcv 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 RsSmcv has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

RsSmcv 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 RsSmcv 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 = RsSmcv('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 RsSmcv 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, RsSmcv 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 RsSmcv object
"""

import threading
from RsSmcv import *


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


driver = RsSmcv('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 RsSmcv objects with shared session
"""

import threading
from RsSmcv import *


def execute(session: RsSmcv, 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 = RsSmcv('TCPIP::192.168.56.101::INSTR')
driver2 = RsSmcv.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 RsSmcv takes care of it for you. The text below describes this scenario.

Run the following example:

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

import threading
from RsSmcv import *


def execute(session: RsSmcv, 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 = RsSmcv('TCPIP::192.168.56.101::INSTR')
driver2 = RsSmcv('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 RsSmcv import *

driver = RsSmcv('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 = RsSmcv('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 RsSmcv import *

driver = RsSmcv('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 RsSmcv import *

driver = RsSmcv('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 All

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

3.3 AmSourceInt

```
# Example value:
value = enums.AmSourceInt.LF1
# All values (6x):
LF1 | LF12 | LF1Noise | LF2 | LF2Noise | NOISe
```

3.4 AnalogDigital

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

3.5 ArbLevMode

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

3.6 ArbMultCarrCresMode

```
# Example value:
value = enums.ArbMultCarrCresMode.MAX
# All values (3x):
MAX | MIN | OFF
```

3.7 ArbMultCarrLevRef

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

3.8 ArbMultCarrSigDurMod

```
# Example value:
value = enums.ArbMultCarrSigDurMod.LCM
# All values (4x):
LCM | LONG | SHORt | USER
```

3.9 ArbMultCarrSpacMode

```
# Example value:
value = enums.ArbMultCarrSpacMode.ARBitrary
# All values (2x):
ARBitrary | EQUidistant
```

3.10 ArbSegmNextSource

```
# Example value:
value = enums.ArbSegmNextSource.INTernal
# All values (2x):
INTernal | NSEGM1
```

3.11 ArbSignType

```
# Example value:
value = enums.ArbSignType.AWGN
# All values (4x):
AWGN | CIQ | RECT | SINE
```

3.12 ArbTrigSegmModeNoEhop

```
# Example value:
value = enums.ArbTrigSegmModeNoEhop.NEXT
# All values (4x):
NEXT | NSEam | SAME | SEQuencer
```

3.13 ArbWaveSegmClocMode

```
# Example value:
value = enums.ArbWaveSegmClocMode.HIGHest
# All values (3x):
HIGHest | UNCHanged | USER
```

3.14 ArbWaveSegmMarkMode

```
# Example value:
value = enums.ArbWaveSegmMarkMode.IGNore
# All values (2x):
IGNore | TAKE
```

3.15 ArbWaveSegmPowMode

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

3.16 ArbWaveSegmRest

```
# Example value:
value = enums.ArbWaveSegmRest.MRK1
# All values (5x):
MRK1 | MRK2 | MRK3 | MRK4 | OFF
```

3.17 Atsc30Coderate

```
# First value:
value = enums.Atsc30Coderate.R10_15
# Last value:
value = enums.Atsc30Coderate.R9_15
# All values (12x):
R10_15 | R11_15 | R12_15 | R13_15 | R2_15 | R3_15 | R4_15 | R5_15
R6_15 | R7_15 | R8_15 | R9_15
```

3.18 Atsc30Constellation

```
# Example value:
value = enums.Atsc30Constellation.T1024
# All values (6x):
T1024 | T16 | T256 | T4 | T4096 | T64
```

3.19 Atsc30Depth

```
# Example value:
value = enums.Atsc30Depth.D1024
# All values (6x):
D1024 | D1254 | D1448 | D512 | D724 | D887
```

3.20 Atsc30EmergencyAlertSignaling

```
# Example value:
value = enums.Atsc30EmergencyAlertSignaling.NOEMergency
# All values (4x):
NOEMergency | SET1 | SET2 | SET3
```

3.21 Atsc30FecType

```
# Example value:
value = enums.Atsc30FecType.B16K
# All values (6x):
B16K | B64K | C16K | C64K | O16K | O64K
```

3.22 Atsc30FftSize

```
# Example value:
value = enums.Atsc30FftSize.M16K
# All values (3x):
M16K | M32K | M8K
```

3.23 Atsc30FrameInfoBandwidth

```
# Example value:
value = enums.Atsc30FrameInfoBandwidth.BW_6
# All values (4x):
BW_6 | BW_7 | BW_8 | BW8G
```

3.24 Atsc30FrameLengthMode

```
# Example value:
value = enums.Atsc30FrameLengthMode.SYMBol
# All values (2x):
SYMBol | TIME
```

3.25 Atsc30GuardInterval

```
# First value:
value = enums.Atsc30GuardInterval.G1024
# Last value:
value = enums.Atsc30GuardInterval.G768
# All values (12x):
G1024 | G1536 | G192 | G2048 | G2432 | G3072 | G3648 | G384
G4096 | G4864 | G512 | G768
```

3.26 Atsc30InputSignalTestSignal

```
# Example value:
value = enums.Atsc30InputSignalTestSignal.TIPP
# All values (2x):
TIPP | TTSP
```

3.27 Atsc30InputType

```
# Example value:
value = enums.Atsc30InputType.IP
# All values (2x):
IP | TS
```

3.28 Atsc30L1DetailAdditionalParityMode

```
# Example value:
value = enums.Atsc30L1DetailAdditionalParityMode.K1
# All values (3x):
K1 | K2 | OFF
```

3.29 Atsc30Layer

```
# Example value:
value = enums.Atsc30Layer.CORE
# All values (2x):
CORE | ENHanced
```

3.30 Atsc30LdmInjectionLayer

```
# First value:
value = enums.Atsc30LdmInjectionLayer.L00
# Last value:
value = enums.Atsc30LdmInjectionLayer.L90
# All values (31x):
L00 | L05 | L10 | L100 | L110 | L120 | L130 | L140
L15 | L150 | L160 | L170 | L180 | L190 | L20 | L200
L210 | L220 | L230 | L240 | L25 | L250 | L30 | L35
L40 | L45 | L50 | L60 | L70 | L80 | L90
```

3.31 Atsc30LowLevelSignaling

```
# Example value:
value = enums.Atsc30LowLevelSignaling.ABSent
# All values (2x):
ABSent | PRESent
```

3.32 Atsc30MinTimeToNext

```
# First value:
value = enums.Atsc30MinTimeToNext.N100
# Last value:
value = enums.Atsc30MinTimeToNext.NOTapplicable
# All values (33x):
N100 | N1000 | N1100 | N1200 | N1400 | N150 | N1500 | N1600
N1700 | N1900 | N200 | N2100 | N2300 | N250 | N2500 | N2700
N2900 | N300 | N3300 | N350 | N3700 | N400 | N4100 | N4500
N4900 | N50 | N500 | N5300 | N600 | N700 | N800 | N900
NOTapplicable
```

3.33 Atsc30Miso

```
# Example value:
value = enums.Atsc30Miso.C256
# All values (3x):
C256 | C64 | OFF
```

3.34 Atsc30PilotPattern

```
# Example value:
value = enums.Atsc30PilotPattern.D12
# All values (8x):
D12 | D16 | D24 | D3 | D32 | D4 | D6 | D8
```

3.35 Atsc30PilotPatternSiso

```
# First value:
value = enums.Atsc30PilotPatternSiso.SP12_2
# Last value:
value = enums.Atsc30PilotPatternSiso.SP8_4
# All values (16x):
SP12_2 | SP12_4 | SP16_2 | SP16_4 | SP24_2 | SP24_4 | SP3_2 | SP3_4
SP32_2 | SP32_4 | SP4_2 | SP4_4 | SP6_2 | SP6_4 | SP8_2 | SP8_4
```

3.36 Atsc30Protocol

```
# Example value:
value = enums.Atsc30Protocol.AUTO
# All values (3x):
AUTO | RTP | UDP
```

3.37 Atsc30TestIppAcket

```
# Example value:
value = enums.Atsc30TestIppAcket.HUDP
# All values (1x):
HUDP
```

3.38 Atsc30TimeInfo

```
# Example value:
value = enums.Atsc30TimeInfo.MSEC
# All values (4x):
MSEC | NSEC | OFF | USEC
```

3.39 Atsc30TimeInfoL1BasicFecType

```
# Example value:
value = enums.Atsc30TimeInfoL1BasicFecType.MOD1
# All values (7x):
MOD1 | MOD2 | MOD3 | MOD4 | MOD5 | MOD6 | MOD7
```

3.40 Atsc30TimeInterleaverMode

```
# Example value:
value = enums.Atsc30TimeInterleaverMode.CTI
# All values (3x):
CTI | HTI | OFF
```

3.41 Atsc30TxIdInjectionLevel

```
# First value:
value = enums.Atsc30TxIdInjectionLevel.L120
# Last value:
value = enums.Atsc30TxIdInjectionLevel.OFF
# All values (14x):
L120 | L150 | L180 | L210 | L240 | L270 | L300 | L330
L360 | L390 | L420 | L450 | L90 | OFF
```

3.42 Atsc30TxIdMode

```
# Example value:
value = enums.Atsc30TxIdMode.AUTo
# All values (3x):
AUTo | MANual | OFF
```

3.43 Atsc30Type

```
# Example value:
value = enums.Atsc30Type.DISPersed
# All values (2x):
DISPersed | NONDispersed
```

3.44 AtscmhBuryRatio

```
# Example value:
value = enums.AtscmhBuryRatio.DB21
# All values (7x):
DB21 | DB24 | DB27 | DB30 | DB33 | DB36 | DB39
```

3.45 AtscmhCodingConstel

```
# Example value:
value = enums.AtscmhCodingConstel.VSB8
# All values (1x):
VSB8
```

3.46 AtscmhCodingInputSignalPacketLength

```
# Example value:
value = enums.AtscmhCodingInputSignalPacketLength.INValid
# All values (3x):
INValid | P188 | P208
```

3.47 AtscmhCodingRolloff

```
# Example value:
value = enums.AtscmhCodingRolloff.R115
# All values (1x):
R115
```

3.48 AtscmhGeneralVsbFrequency

```
# Example value:
value = enums.AtscmhGeneralVsbFrequency.CENTer
# All values (2x):
CENTer | PILot
```

3.49 AtscmhInputSignalTestSignal

```
# Example value:
value = enums.AtscmhInputSignalTestSignal.PBEM
# All values (4x):
PBEM | PBET | PBIN | TTSP
```

3.50 AudioBcFmDarcInformation

```
# Example value:
value = enums.AudioBcFmDarcInformation.DATa
# All values (3x):
DATa | OFF | PRBS
```

3.51 AudioBcFmInputSignalAfMode

```
# Example value:
value = enums.AudioBcFmInputSignalAfMode.LEFT
# All values (5x):
LEFT | RELeft | REMLeft | RIGHt | RNELeft
```

3.52 AudioBcFmModulationMode

```
# Example value:
value = enums.AudioBcFmModulationMode.MONO
# All values (2x):
MONO | STEReo
```

3.53 AudioBcFmModulationPreemphasis

```
# Example value:
value = enums.AudioBcFmModulationPreemphasis.D50us
# All values (3x):
D50us | D75us | OFF
```

3.54 AudioBcInputSignal

```
# Example value:
value = enums.AudioBcInputSignal.AGENerator
# All values (4x):
AGENerator | APLayer | EXTernal | OFF
```

3.55 AutoManStep

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

3.56 AutoManualMode

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

3.57 AutoStep

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

3.58 AutoUser

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

3.59 BasebandModShape

```
# Example value:
value = enums.BasebandModShape.SINE
# All values (1x):
SINE
```

3.60 BasebandPulseMode

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

3.61 BasebandPulseTransType

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

3.62 BbCodMode

```
# Example value:
value = enums.BbCodMode.BBIN
# All values (2x):
BBIN | CODer
```

3.63 BbConfig

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

3.64 BbDigInpBb

```
# First value:
value = enums.BbDigInpBb.A
# Last value:
value = enums.BbDigInpBb.NONE
# All values (9x):
A | B | C | D | E | F | G | H
NONE
```

3.65 BbImpOptMode

```
# Example value:
value = enums.BbImpOptMode.FAST
# All values (4x):
FAST | QHIGh | QHTable | UCORrection
```

3.66 BbinInterfaceMode

```
# Example value:
value = enums.BbinInterfaceMode.DIGital
# All values (2x):
DIGital | HSDin
```

3.67 BbinSampRateModeb

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

3.68 BboutClocSour

```
# Example value:
value = enums.BboutClocSour.DIN
# All values (3x):
DIN | DOUT | USER
```

3.69 BcInputSignalSource

```
# Example value:
value = enums.BcInputSignalSource.SPDif
# All values (1x):
SPDif
```

3.70 BicmFecFrame

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

3.71 ByteOrder

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

3.72 CalDataMode

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

3.73 CalDataUpdate

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

3.74 CalPowAttMode

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

3.75 CfrAlgo

```
# Example value:
value = enums.CfrAlgo.CLFiltering
# All values (2x):
CLFiltering | PCANcellation
```

3.76 CfrFiltMode

```
# Example value:
value = enums.CfrFiltMode.ENHanced
# All values (2x):
ENHanced | SIMPle
```

3.77 ClockModeUnits

```
# Example value:
value = enums.ClockModeUnits.MSAMple
# All values (2x):
MSAMple | SAMPle
```

3.78 ClockSourceB

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

3.79 ClocOutpMode

```
# Example value:
value = enums.ClocOutpMode.BIT
# All values (2x):
BIT | SYMBol
```

3.80 ClocSourBb

```
# Example value:
value = enums.ClocSourBb.AINTernal
# All values (4x):
AINTernal | COUPled | EXTernal | INTernal
```

3.81 ClocSyncModeSgt

```
# Example value:
value = enums.ClocSyncModeSgt.DIIN
# All values (4x):
DIIN | NONE | PRIMary | SECondary
```

3.82 CodingChannelBandwidth

```
# Example value:
value = enums.CodingChannelBandwidth.BW_6
# All values (3x):
BW_6 | BW_7 | BW_8
```

3.83 CodingCoderate

```
# Example value:
value = enums.CodingCoderate.R1_2
# All values (5x):
R1_2 | R2_3 | R3_4 | R5_6 | R7_8
```

3.84 CodingGuardInterval

```
# Example value:
value = enums.CodingGuardInterval.G1_16
# All values (4x):
G1_16 | G1_32 | G1_4 | G1_8
```

3.85 CodingInputFormat

```
# Example value:
value = enums.CodingInputFormat.ASI
# All values (2x):
ASI | SMPTE
```

3.86 CodingInputSignalInputA

```
# Example value:
value = enums.CodingInputSignalInputA.ASI1
# All values (8x):
ASI1 | ASI2 | IP | SMP1 | SMP2 | TS | TS1 | TS2
```

3.87 CodingInputSignalInputAsi

```
# Example value:
value = enums.CodingInputSignalInputAsi.ASI1
# All values (4x):
ASI1 | ASI2 | ASIFront | ASIRear
```

3.88 CodingInputSignalInputB

```
# Example value:
value = enums.CodingInputSignalInputB.ASIFront
# All values (6x):
ASIFront | ASIRear | IP | SPIFront | SPIRear | TS
```

3.89 CodingInputSignalInputSfe

```
# Example value:
value = enums.CodingInputSignalInputSfe.ASI1
# All values (8x):
ASI1 | ASI2 | ASIFront | ASIRear | SMP1 | SMP2 | SMPFront | SMPRear
```

3.90 CodingInputSignalPacketLength

```
# Example value:
value = enums.CodingInputSignalPacketLength.INValid
# All values (2x):
INValid | P188
```

3.91 CodingInputSignalSource

```
# Example value:
value = enums.CodingInputSignalSource.EXTernal
# All values (3x):
EXTernal | TESTsignal | TSPLayer
```

3.92 CodingInputSignalTestSignal

```
# Example value:
value = enums.CodingInputSignalTestSignal.TTSP
# All values (1x):
TTSP
```

3.93 CodingIpType

```
# Example value:
value = enums.CodingIpType.MULTicast
# All values (2x):
MULTicast | UNIcast
```

3.94 CodingIsdbtCodingConstel

```
# Example value:
value = enums.CodingIsdbtCodingConstel.C_16QAM
# All values (4x):
C_16QAM | C_64QAM | C_DQPSK | C_QPSK
```

3.95 CodingIsdbtMode

```
# Example value:
value = enums.CodingIsdbtMode.M1_2K
# All values (3x):
M1_2K | M2_4K | M3_8K
```

3.96 CodingPacketLength

```
# Example value:
value = enums.CodingPacketLength.INV
# All values (4x):
INV | P188 | P204 | P208
```

3.97 CodingPortions

```
# Example value:
value = enums.CodingPortions.CCC
# All values (7x):
CCC | DCC | DDC | DDD | PCC | PDC | PDD
```

3.98 CodingTimeInterleaving

```
# Example value:
value = enums.CodingTimeInterleaving._0
# All values (7x):
_0 | _1 | _16 | _2 | _32 | _4 | _8
```

3.99 Colour

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

3.100 ConnDirection

```
# Example value:
value = enums.ConnDirection.INPut
# All values (3x):
INPut | OUTPut | UNUSed
```

3.101 Count

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

3.102 DabDataSour

```
# Example value:
value = enums.DabDataSour.ALL0
# All values (5x):
ALL0 | ALL1 | ETI | PN15 | PN23
```

3.103 DabTxMode

```
# Example value:
value = enums.DabTxMode.I
# All values (4x):
I | II | III | IV
```

3.104 DetAtt

```
# Example value:
value = enums.DetAtt.HIGH
# All values (5x):
HIGH | LOW | MED | OFF | OVR
```

3.105 DevExpFormat

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

3.106 DexchExtension

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

3.107 DexchMode

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

3.108 DexchSepCol

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

3.109 DexchSepDec

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

3.110 DispKeybLockMode

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

3.111 DmFilterA

```
# First value:
value = enums.DmFilterA.APCO25
# Last value:
value = enums.DmFilterA.SPHase
# All values (18x):
APCO25 | C2K3x | COEQualizer | COF705 | COFequalizer | CONE | COSine | DIRac
ENPShape | EWPShape | GAUSs | LGAuss | LPASs | LPASSEVM | PGAuss | RCOSine
RECTangle | SPHase
```

3.112 DmTrigMode

```
# Example value:
value = enums.DmTrigMode.AAUTo
# All values (5x):
AAUTo | ARETrigger | AUTO | RETRigger | SINGle
```

3.113 DpdPowRef

```
# Example value:
value = enums.DpdPowRef.ADPD
# All values (3x):
ADPD | BDPD | SDPD
```

3.114 DpdShapeMode

```
# Example value:
value = enums.DpdShapeMode.NORMalized
# All values (3x):
NORMalized | POLYnomial | TABLe
```

3.115 DrmCodingChannelBw

```
# Example value:
value = enums.DrmCodingChannelBw.INV
# All values (8x):
INV | K045 | K05 | K09 | K10 | K100 | K18 | K20
```

3.116 DrmCodingCoderate

```
# First value:
value = enums.DrmCodingCoderate.INV
# Last value:
value = enums.DrmCodingCoderate.R078
# All values (17x):
INV | R025 | R033 | R040 | R041 | R045 | R048 | R050
R055 | R057 | R058 | R060 | R062 | R066 | R071 | R072
R078
```

3.117 DrmCodingConstelMsc

```
# Example value:
value = enums.DrmCodingConstelMsc.INV
# All values (6x):
INV | Q16 | Q4 | Q64I | Q64N | Q64Q
```

3.118 DrmCodingConstelSdc

```
# Example value:
value = enums.DrmCodingConstelSdc.INV
# All values (3x):
INV | Q16 | Q4
```

3.119 DrmCodingInterleaver

```
# Example value:
value = enums.DrmCodingInterleaver.INV
# All values (4x):
INV | MS4 | MS6 | S2
```

3.120 DrmCodingProtectionLevelMsc

```
# Example value:
value = enums.DrmCodingProtectionLevelMsc._0
# All values (5x):
_0 | _1 | _2 | _3 | INV
```

3.121 DrmCodingProtectionLevelSdc

```
# Example value:
value = enums.DrmCodingProtectionLevelSdc._0
# All values (3x):
_0 | _1 | INV
```

3.122 DrmCodingProtectionProfileMsc

```
# Example value:
value = enums.DrmCodingProtectionProfileMsc.HPP
# All values (4x):
HPP | INV | LPP | VSPP
```

3.123 DrmCodingProtectionProfileSdc

```
# Example value:
value = enums.DrmCodingProtectionProfileSdc.EEP
# All values (2x):
EEP | INV
```

3.124 DrmCodingRobustness

```
# Example value:
value = enums.DrmCodingRobustness.A
# All values (6x):
A | B | C | D | E | INV
```

3.125 DrmInputSignalLayerType

```
# Example value:
value = enums.DrmInputSignalLayerType.BASE
# All values (3x):
BASE | ENHancement | INV
```

3.126 DrmInputSignalServices

```
# Example value:
value = enums.DrmInputSignalServices._0
# All values (6x):
_0 | _1 | _2 | _3 | _4 | INV
```

3.127 DrmInputSignalSource

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

3.128 DtmbCodingCoderate

```
# Example value:
value = enums.DtmbCodingCoderate.R04
# All values (3x):
R04 | R06 | R08
```

3.129 DtmbCodingConstel

```
# Example value:
value = enums.DtmbCodingConstel.D16
# All values (5x):
D16 | D32 | D4 | D4NR | D64
```

3.130 DtmbCodingGipN

```
# Example value:
value = enums.DtmbCodingGipN.CONSt
# All values (2x):
CONSt | VAR
```

3.131 DtmbCodingGuardInterval

```
# Example value:
value = enums.DtmbCodingGuardInterval.G420
# All values (3x):
G420 | G595 | G945
```

3.132 DtmbCodingInputSignalPacketLength

```
# Example value:
value = enums.DtmbCodingInputSignalPacketLength.INV
# All values (2x):
INV | P188
```

3.133 DtmbCodingTimeInterleaver

```
# Example value:
value = enums.DtmbCodingTimeInterleaver.I240
# All values (3x):
I240 | I720 | OFF
```

3.134 DvbcCodingDvbcCodingConstel

```
# Example value:
value = enums.DvbcCodingDvbcCodingConstel.C128
# All values (5x):
C128 | C16 | C256 | C32 | C64
```

3.135 DvbcCodingDvbcCodingRolloff

```
# Example value:
value = enums.DvbcCodingDvbcCodingRolloff._0_dot_13
# All values (2x):
_0_dot_13 | _0_dot_15
```

3.136 DvbcCodingDvbcInputSignalTestSignal

```
# Example value:
value = enums.DvbcCodingDvbcInputSignalTestSignal.PBDE
# All values (3x):
PBDE | PBEM | TTSP
```

3.137 Dvbs2CodingCoderateSfe

```
# First value:
value = enums.Dvbs2CodingCoderateSfe.R1_2
# Last value:
value = enums.Dvbs2CodingCoderateSfe.UNDef
# All values (12x):
R1_2 | R1_3 | R1_4 | R2_3 | R2_5 | R3_4 | R3_5 | R4_5
R5_6 | R8_9 | R9_10 | UNDef
```

3.138 Dvbs2CodingConstelSfe

```
# Example value:
value = enums.Dvbs2CodingConstelSfe.A16
# All values (5x):
A16 | A32 | S4 | S8 | UNDef
```

3.139 Dvbs2CodingFecFrame

```
# Example value:
value = enums.Dvbs2CodingFecFrame.MEDium
# All values (3x):
MEDium | NORMal | SHORt
```

3.140 Dvbs2CodingModCod

```
# First value:
value = enums.Dvbs2CodingModCod._0
# Last value:
value = enums.Dvbs2CodingModCod._99
# All values (107x):
_0 | _1 | _10 | _100 | _101 | _102 | _103 | _104
_105 | _106 | _11 | _12 | _13 | _14 | _15 | _16
_17 | _18 | _19 | _2 | _20 | _21 | _22 | _23
_24 | _25 | _26 | _27 | _28 | _29 | _3 | _30
_31 | _32 | _33 | _34 | _35 | _36 | _37 | _38
_39 | _4 | _40 | _41 | _42 | _43 | _44 | _45
_46 | _47 | _48 | _49 | _5 | _50 | _51 | _52
_53 | _54 | _55 | _56 | _57 | _58 | _59 | _6
_60 | _61 | _62 | _63 | _64 | _65 | _66 | _67
_68 | _69 | _7 | _70 | _71 | _72 | _73 | _74
_75 | _76 | _77 | _78 | _79 | _8 | _80 | _81
_82 | _83 | _84 | _85 | _86 | _87 | _88 | _89
_9 | _90 | _91 | _92 | _93 | _94 | _95 | _96
_97 | _98 | _99
```

3.141 Dvbs2CodingRolloff

```
# Example value:
value = enums.Dvbs2CodingRolloff._0_dot_05
# All values (6x):
_0_dot_05 | _0_dot_10 | _0_dot_15 | _0_dot_20 | _0_dot_25 | _0_dot_35
```

3.142 Dvbs2InputSignalCmMode

```
# Example value:
value = enums.Dvbs2InputSignalCmMode.ACM
# All values (3x):
ACM | CCM | VCM
```

3.143 Dvbs2InputSignalTestSignal

```
# Example value:
value = enums.Dvbs2InputSignalTestSignal.TGSP
# All values (2x):
TGSP | TTSP
```

3.144 DvbsCodingDvbsCodingCoderate

```
# Example value:
value = enums.DvbsCodingDvbsCodingCoderate.R1_2
# All values (6x):
R1_2 | R2_3 | R3_4 | R5_6 | R7_8 | R8_9
```

3.145 DvbsCodingDvbsCodingConstel

```
# Example value:
value = enums.DvbsCodingDvbsCodingConstel.S16
# All values (3x):
S16 | S4 | S8
```

3.146 DvbsCodingDvbsCodingRolloff

```
# Example value:
value = enums.DvbsCodingDvbsCodingRolloff._0_dot_20
# All values (3x):
_0_dot_20 | _0_dot_25 | _0_dot_35
```

3.147 DvbsCodingDvbsInputSignalTestSignal

```
# Example value:
value = enums.DvbsCodingDvbsInputSignalTestSignal.PBEC
# All values (2x):
PBEC | TTSP
```

3.148 Dvbt2BicmCoderate

```
# Example value:
value = enums.Dvbt2BicmCoderate.R1_2
# All values (8x):
R1_2 | R1_3 | R2_3 | R2_5 | R3_4 | R3_5 | R4_5 | R5_6
```

3.149 Dvbt2BicmConstel

```
# Example value:
value = enums.Dvbt2BicmConstel.T16
# All values (4x):
T16 | T256 | T4 | T64
```

3.150 Dvbt2FramingChannelBandwidth

```
# Example value:
value = enums.Dvbt2FramingChannelBandwidth.BW_2
# All values (5x):
BW_2 | BW_5 | BW_6 | BW_7 | BW_8
```

3.151 Dvbt2FramingFftSize

```
# First value:
value = enums.Dvbt2FramingFftSize.M16E
# Last value:
value = enums.Dvbt2FramingFftSize.M8K
# All values (9x):
M16E | M16K | M1K | M2K | M32E | M32K | M4K | M8E
M8K
```

3.152 Dvbt2FramingGuardInterval

```
# Example value:
value = enums.Dvbt2FramingGuardInterval.G1_16
# All values (7x):
G1_16 | G1_32 | G1_4 | G1_8 | G1128 | G19128 | G19256
```

3.153 Dvbt2FramingPilotPattern

```
# Example value:
value = enums.Dvbt2FramingPilotPattern.PP1
# All values (8x):
PP1 | PP2 | PP3 | PP4 | PP5 | PP6 | PP7 | PP8
```

3.154 Dvbt2InputIssy

```
# Example value:
value = enums.Dvbt2InputIssy.LONG
# All values (3x):
LONG | OFF | SHORt
```

3.155 Dvbt2InputSignalCm

```
# Example value:
value = enums.Dvbt2InputSignalCm.ACM
# All values (2x):
ACM | CCM
```

3.156 Dvbt2InputSignalMeasurementMode

```
# Example value:
value = enums.Dvbt2InputSignalMeasurementMode.ABSOLUTE
# All values (2x):
ABSOLUTE | DELTA
```

3.157 Dvbt2ModeStreamAdapterPlpType

```
# Example value:
value = enums.Dvbt2ModeStreamAdapterPlpType.COMMon
# All values (3x):
COMMon | DT1 | DT2
```

3.158 Dvbt2PlpInputFormat

```
# Example value:
value = enums.Dvbt2PlpInputFormat.GCS
# All values (4x):
GCS | GFPS | GSE | TS
```

3.159 Dvbt2T2SystemFefPayload

```
# Example value:
value = enums.Dvbt2T2SystemFefPayload.NOISe
# All values (2x):
NOISe | NULL
```

3.160 Dvbt2T2SystemL1PostModulation

```
# Example value:
value = enums.Dvbt2T2SystemL1PostModulation.T16
# All values (4x):
T16 | T2 | T4 | T64
```

3.161 Dvbt2T2SystemL1T2Version

```
# Example value:
value = enums.Dvbt2T2SystemL1T2Version.V111
# All values (3x):
V111 | V121 | V131
```

3.162 Dvbt2T2SystemMisoGroupScpi

```
# Example value:
value = enums.Dvbt2T2SystemMisoGroupScpi.G1
# All values (2x):
G1 | G2
```

3.163 Dvbt2T2SystemProfileMode

```
# Example value:
value = enums.Dvbt2T2SystemProfileMode.MULTI
# All values (2x):
MULTI | SINGLE
```

3.164 Dvbt2Transmission

```
# Example value:
value = enums.Dvbt2Transmission.MISO
# All values (5x):
MISO | NONT2 | SISO | T2LM | T2LS
```

3.165 DvbtCodingChannelBandwidth

```
# Example value:
value = enums.DvbtCodingChannelBandwidth.BW_5
# All values (5x):
BW_5 | BW_6 | BW_7 | BW_8 | BW_Var
```

3.166 DvbtCodingConstel

```
# Example value:
value = enums.DvbtCodingConstel.T16
# All values (3x):
T16 | T4 | T64
```

3.167 DvbtCodingDvbhSymbolInterleaver

```
# Example value:
value = enums.DvbtCodingDvbhSymbolInterleaver.INDepth
# All values (2x):
INDepth | NATive
```

3.168 DvbtCodingFftMode

```
# Example value:
value = enums.DvbtCodingFftMode.M2K
# All values (3x):
M2K | M4K | M8K
```

3.169 DvbtCodingGuardInterval

```
# Example value:
value = enums.DvbtCodingGuardInterval.G1
# All values (5x):
G1 | G1_16 | G1_32 | G1_4 | G1_8
```

3.170 DvbtCodingHierarchy

```
# Example value:
value = enums.DvbtCodingHierarchy.A1
# All values (4x):
A1 | A2 | A4 | NONHier
```

3.171 DvbxCodingInputSignalPacketLength

```
# Example value:
value = enums.DvbxCodingInputSignalPacketLength.INValid
# All values (3x):
INValid | P188 | P204
```

3.172 EmulSgtBbSystemConfiguration

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

3.173 EmulSgtPowLevBehaviour

```
# Example value:
value = enums.EmulSgtPowLevBehaviour.AUTO
# All values (6x):
AUTO | CVSWr | CWSWr | MONotone | UNINterrupted | USER
```

3.174 EmulSgtRefLoOutput

```
# Example value:
value = enums.EmulSgtRefLoOutput.LO
# All values (3x):
LO | OFF | REF
```

3.175 EmulSgtRoscOutputFreq

```
# Example value:
value = enums.EmulSgtRoscOutputFreq._1000MHZ
# All values (4x):
_1000MHZ | _100MHZ | _10MHZ | _13MHZ
```

3.176 EnetworkMode

```
# Example value:
value = enums.EnetworkMode.MFN
# All values (2x):
MFN | SFN
```

3.177 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
```

3.178 FilterWidth

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

3.179 FormData

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

3.180 FormStatReg

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

3.181 FreqMode

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

3.182 FreqStepMode

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

3.183 HardCopyImageFormat

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

3.184 HardCopyRegion

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

3.185 IecTermMode

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

3.186 ImpG50G1KcoerceG10K

```
# Example value:
value = enums.ImpG50G1KcoerceG10K.G1K
# All values (2x):
G1K | G50
```

3.187 InclExcl

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

3.188 InpOutpConnGlbMapSignb

```
# First value:
value = enums.InpOutpConnGlbMapSignb.CLOCK1
# Last value:
value = enums.InpOutpConnGlbMapSignb.TS
# All values (11x):
CLOCK1 | ETI | INSTtrigger | MARKA1 | NONE | NSEGM1 | PPS | SDIF
SYNCIN | TRIG1 | TS
```

3.189 InputImpRf

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

3.190 InputSignalPacketLength

```
# Example value:
value = enums.InputSignalPacketLength.INValid
# All values (4x):
INValid | P188 | P204 | P208
```

3.191 InputSignalTestSignal

```
# Example value:
value = enums.InputSignalTestSignal.PAFC
# All values (3x):
PAFC | PBEC | TTSP
```

3.192 IqGain

```
# Example value:
value = enums.IqGain.DB0
# All values (7x):
DB0 | DB2 | DB4 | DB6 | DB8 | DBM2 | DBM4
```

3.193 IqMode

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

3.194 IqOutDispViaType

```
# Example value:
value = enums.IqOutDispViaType.LEVel
# All values (2x):
LEVel | PEP
```

3.195 IqOutEnvAdaption

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

3.196 IqOutEnvDetrFunc

```
# Example value:
value = enums.IqOutEnvDetrFunc.F1
# All values (3x):
F1 | F2 | F3
```

3.197 IqOutEnvEtRak

```
# Example value:
value = enums.IqOutEnvEtRak.ET1V2
# All values (4x):
ET1V2 | ET1V5 | ET2V0 | USER
```

3.198 IqOutEnvInterp

```
# Example value:
value = enums.IqOutEnvInterp.LINear
# All values (3x):
LINear | OFF | POWer
```

3.199 IqOutEnvScale

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

3.200 IqOutEnvShapeMode

```
# Example value:
value = enums.IqOutEnvShapeMode.DETRoughing
# All values (6x):
DETRoughing | LINear | OFF | POLYnomial | POWer | TABLe
```

3.201 IqOutEnvTerm

```
# Example value:
value = enums.IqOutEnvTerm.GROund
# All values (2x):
GROund | WIRE
```

3.202 IqOutEnvVrEf

```
# Example value:
value = enums.IqOutEnvVrEf.VCC
# All values (2x):
VCC | VOUT
```

3.203 IqOutMode

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

3.204 IqOutType

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

3.205 IsdbtCodingSystem

```
# Example value:
value = enums.IsdbtCodingSystem.T
# All values (3x):
T | TSB1 | TSB3
```

3.206 IsdbtEewInfoType

```
# Example value:
value = enums.IsdbtEewInfoType.CANCeled
# All values (2x):
CANCeled | ISSued
```

3.207 IsdbtEewSignalType

```
# Example value:
value = enums.IsdbtEewSignalType.TWA
# All values (4x):
TWA | TWOA | WWA | WWOA
```

3.208 IsdbtSpecialTmcc

```
# Example value:
value = enums.IsdbtSpecialTmcc.CURRent
# All values (2x):
CURRent | UNUSed
```

3.209 IsdbtSpecialTxParam

```
# First value:
value = enums.IsdbtSpecialTxParam.N1
# Last value:
value = enums.IsdbtSpecialTxParam.NORMal
# All values (15x):
N1 | N10 | N11 | N12 | N13 | N14 | N15 | N2
N4 | N5 | N6 | N7 | N8 | N9 | NORMal
```

3.210 J83BcodingJ83BcodingConstel

```
# Example value:
value = enums.J83BcodingJ83BcodingConstel.J1024
# All values (3x):
J1024 | J256 | J64
```

3.211 J83BcodingJ83BcodingRolloff

```
# Example value:
value = enums.J83BcodingJ83BcodingRolloff._0_dot_12
# All values (2x):
_0_dot_12 | _0_dot_18
```

3.212 J83BcodingJ83BinputSignalTestSignal

```
# Example value:
value = enums.J83BcodingJ83BinputSignalTestSignal.PBEM
# All values (3x):
PBEM | PBTR | TTSP
```

3.213 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
```

3.214 LfFreqMode

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

3.215 LmodRunMode

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

3.216 LoRaBw

```
# First value:
value = enums.LoRaBw.BW10
# Last value:
value = enums.LoRaBw.BW7
# All values (10x):
BW10 | BW125 | BW15 | BW20 | BW250 | BW31 | BW41 | BW500
BW62 | BW7
```

3.217 LoRaCodRate

```
# Example value:
value = enums.LoRaCodRate.CR0
# All values (5x):
CR0 | CR1 | CR2 | CR3 | CR4
```

3.218 LoRaFreqDfTp

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

3.219 LoRaSf

```
# Example value:
value = enums.LoRaSf.SF10
# All values (7x):
SF10 | SF11 | SF12 | SF6 | SF7 | SF8 | SF9
```

3.220 LoRaSyncMode

```
# Example value:
value = enums.LoRaSyncMode.PRIVate
# All values (2x):
PRIVate | PUBLic
```

3.221 MappingType

```
# Example value:
value = enums.MappingType.A
# All values (2x):
A | B
```

3.222 MarkModeA

```
# Example value:
value = enums.MarkModeA.FRAMe
# All values (6x):
FRAMe | PATTern | PULSe | RATio | RESTart | TRIGger
```

3.223 MultInstMsMode

```
# Example value:
value = enums.MultInstMsMode.PRIMary
# All values (2x):
PRIMary | SECondary
```

3.224 NetMode

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

3.225 NetProtocol

```
# Example value:
value = enums.NetProtocol.TCP
# All values (2x):
TCP | UDP
```

3.226 NetworkMode

```
# Example value:
value = enums.NetworkMode.MFN
# All values (1x):
MFN
```

3.227 NoisAwgnDispMode

```
# Example value:
value = enums.NoisAwgnDispMode.IQOUT1
# All values (2x):
IQOUT1 | RFA
```

3.228 NoisAwgnFseState

```
# Example value:
value = enums.NoisAwgnFseState.ADD
# All values (3x):
ADD | OFF | ONLY
```

3.229 NoisAwgnMode

```
# Example value:
value = enums.NoisAwgnMode.ADD
# All values (3x):
ADD | CW | ONLY
```

3.230 NoisAwgnPowMode

```
# Example value:
value = enums.NoisAwgnPowMode.CN
# All values (3x):
CN | EN | SN
```

3.231 NoisAwgnPowRefMode

```
# Example value:
value = enums.NoisAwgnPowRefMode.CARRier
# All values (2x):
CARRier | NOISe
```

3.232 NormalInverted

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

3.233 NumberA

```
# Example value:
value = enums.NumberA._1
# All values (4x):
_1 | _2 | _3 | _4
```

3.234 OutpConnGlbSignalb

```
# Example value:
value = enums.OutpConnGlbSignalb.MARKA1
# All values (2x):
MARKA1 | NONE
```

3.235 ParameterSetMode

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

3.236 Parity

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

3.237 PathUniCodBbin

```
# Example value:
value = enums.PathUniCodBbin.A
# All values (3x):
A | AB | B
```

3.238 PathUniCodBbinA

```
# Example value:
value = enums.PathUniCodBbinA.A
# All values (1x):
A
```

3.239 PayloadTestStuff

```
# Example value:
value = enums.PayloadTestStuff.H00
# All values (3x):
H00 | HFF | PRBS
```

3.240 PidTestPacket

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

3.241 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
```

3.242 PowAlcDetSensitivityEmulSgt

```
# Example value:
value = enums.PowAlcDetSensitivityEmulSgt.AUTO
# All values (6x):
AUTO | FIXed | HIGH | LOW | MEDium | OFF
```

3.243 PowAlcStateEmulSgt

```
# Example value:
value = enums.PowAlcStateEmulSgt._0
# All values (7x):
_0 | _1 | AUTO | OFF | OFFTable | ON | ONTable
```

3.244 PowAttRfOffMode

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

3.245 PowCntrlSelect

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

3.246 PowerAttMode

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

3.247 PowLevBehaviour

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

3.248 PowSensDisplayPriority

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

3.249 PowSensFiltType

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

3.250 PowSensSource

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

3.251 PulseSoure

```
# Example value:
value = enums.PulseSoure.CODer
# All values (4x):
CODer | EXTernal | INTernal | RANDom
```

3.252 PulsTrigMode

```
# Example value:
value = enums.PulsTrigMode.AUTO
# All values (3x):
AUTO | EGATe | EXTernal
```

3.253 RecScpiCmdMode

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

3.254 RoscFreqExt

```
# Example value:
value = enums.RoscFreqExt._10MHZ
# All values (3x):
_10MHZ | _13MHZ | _5MHZ
```

3.255 RoscOutpFreqMode

```
# Example value:
value = enums.RoscOutpFreqMode.DER10M
# All values (3x):
DER10M | LOOPthrough | OFF
```

3.256 RoscSourSetup

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

3.257 Rs232BdRate

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

3.258 SampRateFifoStatus

```
# Example value:
value = enums.SampRateFifoStatus.OFLow
# All values (3x):
OFLow | OK | URUN
```

3.259 SelftLev

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

3.260 SelftLevWrite

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

3.261 SensorModeAll

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

3.262 SettingsPrbs

```
# Example value:
value = enums.SettingsPrbs.P15_1
# All values (2x):
P15_1 | P23_1
```

3.263 SettingsTestTsPacket

```
# Example value:
value = enums.SettingsTestTsPacket.H184
# All values (2x):
H184 | S187
```

3.264 SfnMode

```
# Example value:
value = enums.SfnMode.ABSolute
# All values (2x):
ABSolute | RELative
```

3.265 SgtUserPlug

```
# First value:
value = enums.SgtUserPlug.CIN
# Last value:
value = enums.SgtUserPlug.TRIGger
# All values (18x):
CIN | COUT | HIGH | LOW | MARRived | MKR1 | MKR2 | MLATency
NEXT | PEMSource | PETRigger | PVOut | SIN | SNValid | SOUT | SVALid
TOUT | TRIGger
```

3.266 SingExtAuto

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

3.267 SlopeType

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

3.268 SourceInt

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

3.269 Spacing

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

3.270 SpecialAcData

```
# Example value:
value = enums.SpecialAcData.ALL1
# All values (2x):
ALL1 | PRBS
```

3.271 StateExtended

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

3.272 StateOn

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

3.273 SweCyclMode

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

3.274 SystConfBbConf

```
# Example value:
value = enums.SystConfBbConf.COUPled
# All values (3x):
COUPled | CPENtity | SEParate
```

3.275 SystConfHsChannels

```
# First value:
value = enums.SystConfHsChannels.CH0
# Last value:
value = enums.SystConfHsChannels.CH8
# All values (9x):
CH0 | CH1 | CH2 | CH3 | CH4 | CH5 | CH6 | CH7
CH8
```

3.276 SystConfOutpMode

```
# Example value:
value = enums.SystConfOutpMode.ALL
# All values (6x):
ALL | ANALog | DIGital | DIGMux | HSALl | HSDigital
```

3.277 SystemPostExtension

```
# Example value:
value = enums.SystemPostExtension.OFF
# All values (1x):
OFF
```

3.278 T2SystemPapr

```
# Example value:
value = enums.T2SystemPapr.OFF
# All values (2x):
OFF | TR
```

3.279 TdmaDataSource

```
# First value:
value = enums.TdmaDataSource.DLISt
# Last value:
value = enums.TdmaDataSource.ZERO
# All values (11x):
DLISt | ONE | PATTern | PN11 | PN15 | PN16 | PN20 | PN21
PN23 | PN9 | ZERO
```

3.280 TdmbInputSignalEtiSignal

```
# Example value:
value = enums.TdmbInputSignalEtiSignal.E537
# All values (4x):
E537 | E559 | ENI | INValid
```

3.281 TdmbInputSignalInputFormat

```
# Example value:
value = enums.TdmbInputSignalInputFormat.ETI
# All values (1x):
ETI
```

3.282 TdmbInputSignalProtectionLevel

```
# First value:
value = enums.TdmbInputSignalProtectionLevel.EP1A
# Last value:
value = enums.TdmbInputSignalProtectionLevel.UP5
# All values (14x):
EP1A | EP1B | EP2A | EP2B | EP3A | EP3B | EP4A | EP4B
UNDefined | UP1 | UP2 | UP3 | UP4 | UP5
```

3.283 TdmbInputSignalProtectionProfile

```
# Example value:
value = enums.TdmbInputSignalProtectionProfile.EEP
# All values (2x):
EEP | UEP
```

3.284 TdmbSettingsPrbs

```
# Example value:
value = enums.TdmbSettingsPrbs.P15_1
# All values (4x):
P15_1 | P20_1 | P23_1 | ZERO
```

3.285 TdmbSpecialTransmissionMode

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

3.286 Test

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

3.287 TestBbGenIqSour

```
# Example value:
value = enums.TestBbGenIqSour.ARB
# All values (4x):
ARB | CONStant | SINE | TTONe
```

3.288 TestExtIqMode

```
# Example value:
value = enums.TestExtIqMode.IQIN
# All values (2x):
IQIN | IQOut
```

3.289 TimeProtocol

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

3.290 TranRecFftLen

```
# Example value:
value = enums.TranRecFftLen.LEN1024
# All values (5x):
LEN1024 | LEN2048 | LEN256 | LEN4096 | LEN512
```

3.291 TranRecMode

```
# Example value:
value = enums.TranRecMode.CCDF
# All values (7x):
CCDF | CONStellation | EYEI | EYEQ | IQ | PSPectrum | VECTor
```

3.292 TranRecSampFactMode

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

3.293 TranRecSize

```
# Example value:
value = enums.TranRecSize.MAXimized
# All values (2x):
MAXimized | MINimized
```

3.294 TranRecSour

```
# Example value:
value = enums.TranRecSour.BBA
# All values (6x):
BBA | BBIA | DO1 | IQO1 | RFA | STRA
```

3.295 TranRecTrigSour

```
# Example value:
value = enums.TranRecTrigSour.MARKer
# All values (2x):
MARKer | SOFTware
```

3.296 TrigDelUnit

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

3.297 TriggerMarkModeA

```
# Example value:
value = enums.TriggerMarkModeA.PATTern
# All values (6x):
PATTern | PULSe | RATio | RESTart | TRIGger | UNCHanged
```

3.298 TriggerSourceB

```
# Example value:
value = enums.TriggerSourceB.BEXTernal
# All values (4x):
BEXTernal | EXTernal | INTernal | OBASeband
```

3.299 TrigRunMode

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

3.300 TrigSour

```
# Example value:
value = enums.TrigSour.BBSY
# All values (4x):
BBSY | EGT1 | EXTernal | INTernal
```

3.301 TrigSweepSourNoHopExtAuto

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

3.302 TspLayerSettingsTestTsPacket

```
# Example value:
value = enums.TspLayerSettingsTestTsPacket.H184
# All values (6x):
H184 | H200 | H204 | S187 | S203 | S207
```

3.303 TspLayerStatus

```
# Example value:
value = enums.TspLayerStatus.PAUSe
# All values (4x):
PAUSe | PLAY | RESet | STOP
```

3.304 TxAudioBcFmRdsAfBorder

```
# Example value:
value = enums.TxAudioBcFmRdsAfBorder.ASC
# All values (2x):
ASC | DESC
```

3.305 TxAudioBcFmRdsEonAfMethod

```
# Example value:
value = enums.TxAudioBcFmRdsEonAfMethod.A
# All values (2x):
A | MAPF
```

3.306 TxAudioBcFmRdsMs

```
# Example value:
value = enums.TxAudioBcFmRdsMs.MUSic
# All values (2x):
MUSic | SPEech
```

3.307 UnchOff

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

3.308 UnitAngle

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

3.309 UnitPower

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

3.310 UnitPowSens

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

3.311 UnitSlB

```
# Example value:
value = enums.UnitSlB.SAMPle
# All values (2x):
SAMPle | SEQuence
```

3.312 UnitSpeed

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

3.313 Unknown

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

3.314 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 AlternaiveFreqList

```
# First value:
value = repcap.AlternaiveFreqList.Nr1
# Range:
Nr1 .. Nr12
# All values (12x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12
```

4.3 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
```

4.4 BlockIdCode

```
# First value:
value = repcap.BlockIdCode.Nr1
# Values (3x):
Nr1 | Nr2 | Nr3
```

4.5 Carrier

```
# First value:
value = repcap.Carrier.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
```

4.6 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
```

4.7 ChannelNull

```
# First value:
value = repcap.ChannelNull.Nr0
# Range:
Nr0 .. Nr63
# All values (64x):
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
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
```

4.8 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
```

4.9 External

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

4.10 GroupTypeVariant

```
# First value:
value = repcap.GroupTypeVariant.Nr1
# Values (2x):
Nr1 | Nr2
```

4.11 Index

```
# First value:
value = repcap.Index.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
```

4.12 InputIx

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

4.13 InputStream

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

4.14 IpVersion

```
# First value:
value = repcap.IpVersion.Nr4
# Values (2x):
Nr4 | Nr6
```

4.15 IqConnector

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

4.16 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
```

4.17 Output

```
# First value:
value = repcap.Output.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
```

4.18 Path

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

4.19 PhysicalLayerPipe

```
# First value:
value = repcap.PhysicalLayerPipe.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
```

4.20 Profile

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

4.21 Stream

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

4.22 SubChannel

```
# First value:
value = repcap.SubChannel.Nr1
# Range:
Nr1 .. Nr32
# All values (32x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
```

(continues on next page)

(continued from previous page)

```
Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23 | Nr24
Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31 | Nr32
```

4.23 Subframe

```
# First value:
value = repcap.Subframe.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
```

4.24 TimeSlice

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

4.25 UserIx

```
# First value:
value = repcap.UserIx.Nr1
# Range:
Nr1 .. Nr48
# All values (48x):
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
```

CHAPTER

FIVE

EXAMPLES

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

```
"""Getting started - how to work with RsSmcv Python package.
This example performs basic RF settings on an SMCV100B instrument.
It shows the RsSmcv calls and their corresponding SCPI commands.
Notice that the python RsSmcv interfaces track the SCPI commands syntax."""

from RsSmcv import *

# Open the session
RsSmcv.assert_minimum_version('5.0.122')
smcv = RsSmcv('TCPIP::10.102.52.52::HISLIP')
# Greetings, stranger...
print(f'Hello, I am: {smcv.utilities.idn_string}')

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

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

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

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

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


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

CHAPTER

SIX

RSSMCV API STRUCTURE

Global RepCaps

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

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

1998 total commands, 24 Subgroups, 4 group commands

Initializes new RsSmcv 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 RsSmcv.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 RsSmcv session.

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

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

Parameters

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

get_ffast() → float

```
# SCPI: FFASt
value: float = driver.get_ffast()
```

No command help available

return
freq: No help available

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_lock() → float

```
# SCPI: LOCK
value: float = driver.get_lock()
```

No command help available

return
value: No help available

get_pfast() → float

```
# SCPI: PFASt
value: float = driver.get_pfast()
```

No command help available

return
power: No help available

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

set_ffast(*freq: float*) → None

```
# SCPI: FFASt
driver.set_ffast(freq = 1.0)
```

No command help available

param freq
No help available

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.

set_pfast(*power: float*) → None

```
# SCPI: PFASt
driver.set_pfast(power = 1.0)
```

No command help available

param power
No help available

unlock(*unlock_id: float*) → None

```
# SCPI: UNLock
driver.unlock(unlock_id = 1.0)
```

No command help available

param unlock_id
No help available

Subgroups

6.1 Calibration

SCPI Command :

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

class CalibrationCls

Calibration commands group definition. 27 total commands, 10 Subgroups, 1 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

Cloning the Group

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

Subgroups

6.1.1 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

6.1.1.1 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.

param force
string

return
measure: 1| ON| 0| OFF

6.1.2 Bbin

SCPI Command :

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

class BbinCls

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

get_measure() → bool

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

No command help available

return
measure: No help available

6.1.3 Data

SCPI Command :

```
CALibration:DATA:EXPort
```

class DataCls

Data commands group definition. 4 total commands, 2 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 RsSmcv.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

6.1.3.1 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

6.1.3.2 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

6.1.3.2.1 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

6.1.3.2.1.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.1.4 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 RsSmcv.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

6.1.4.1 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

6.1.5 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

6.1.6 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

6.1.7 IqModulator

SCPI Commands :

```
CALibration<HW>:IQModulator:FULL
CALibration<HW>:IQModulator:LOCal
```

class IqModulatorCls

IqModulator commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_full() → bool

```
# SCPI: CALibration<HW>:IQModulator:FULL
value: bool = driver.calibration.iqModulator.get_full()
```

No command help available

return
full: No help available

get_local() → bool

```
# SCPI: CALibration<HW>:IQModulator:LOCal
value: bool = driver.calibration.iqModulator.get_local()
```

No command help available

return
local: No help available

Cloning the Group

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

Subgroups

6.1.7.1 Bband

SCPI Command :

```
CALibration:IQModulator:BBANd:[STATe]
```

class BbandCls

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

get_state() → bool

```
# SCPI: CALibration:IQModulator:BBANd:[STATe]
value: bool = driver.calibration.iqModulator.bband.get_state()
```

No command help available

return
modulator: No help available

set_state(*modulator: bool*) → None

```
# SCPI: CALibration:IQModulator:BBANd:[STATe]
driver.calibration.iqModulator.bband.set_state(modulator = False)
```

No command help available

param modulator
No help available

6.1.7.2 IqModulator

SCPI Command :

```
CALibration:IQModulator:IQModulator:[STATe]
```

class IqModulatorCls

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

get_state() → bool

```
# SCPI: CALibration:IQModulator:IQModulator:[STATe]
value: bool = driver.calibration.iqModulator.iqModulator.get_state()
```

No command help available

return
modulator: No help available

set_state(*modulator: bool*) → None

```
# SCPI: CALibration:IQModulator:IQModulator:[STATe]
driver.calibration.iqModulator.iqModulator.set_state(modulator = False)
```

No command help available

param modulator
No help available

6.1.8 Level

SCPI Commands :

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

class LevelCls

Level commands group definition. 6 total commands, 3 Subgroups, 2 group commands

get_det_att() → DetAtt

```
# SCPI: CALibration<HW>:LEVel:DETatt
value: enums.DetAtt = driver.calibration.level.get_det_att()
```

No command help available

return
det_att: 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_det_att(*det_att: DetAtt*) → None

```
# SCPI: CALibration<HW>:LEVel:DETatt
driver.calibration.level.set_det_att(det_att = enums.DetAtt.HIGH)
```

No command help available

param det_att
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

6.1.8.1 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

6.1.8.2 Haccuracy

SCPI Command :

```
CALibration<HW>:LEVel:HACCuracy:[STATe]
```

class HaccuracyCls

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

get_state() → bool

```
# SCPI: CALibration<HW>:LEVel:HACCuracy:[STATe]
value: bool = driver.calibration.level.haccuracy.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: CALibration<HW>:LEVel:HACCuracy:[STATe]
driver.calibration.level.haccuracy.set_state(state = False)
```

No command help available

param state
No help available

6.1.8.3 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

6.1.9 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

6.1.10 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

6.1.10.1 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()
```

Sets a user-defined calibration value for the internal reference frequency.

return
data: integer 0 = the factory value is used (i.e. user-defined calibration is disabled) Value 0: positive calibration values are used Value 0: negative calibration values are used Range: depends on other values

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)
```

Sets a user-defined calibration value for the internal reference frequency.

param data
integer 0 = the factory value is used (i.e. user-defined calibration is disabled) Value 0: positive calibration values are used Value 0: negative calibration values are used Range: depends on other values

6.1.10.2 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 RsSmcv.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 Clock

class ClockCls

Clock 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.clock.clone()
```

Subgroups

6.2.1 InputPy

SCPI Command :

```
CLOCk:INPut:FREQuency
```

class InputPyCls

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

get_frequency() → float

```
# SCPI: CLOCk:INPut:FREQuency
value: float = driver.clock.inputPy.get_frequency()
```

No command help available

return
frequency: No help available

6.2.2 Output

SCPI Command :

```
CLOCk:OUTPut:MODE
```

class OutputCls

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

get_mode() → ClocOutpMode

```
# SCPI: CLOCk:OUTPut:MODE
value: enums.ClocOutpMode = driver.clock.output.get_mode()
```

No command help available

return
mode: No help available

set_mode(*mode: ClocOutpMode*) → None

```
# SCPI: CLOCk:OUTPut:MODE
driver.clock.output.set_mode(mode = enums.ClocOutpMode.BIT)
```

No command help available

param mode
No help available

6.2.3 Sync

SCPI Command :

```
CLOCk:SYNC:[STATe]
```

class SyncCls

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

get_state() → bool

```
# SCPI: CLOCk:SYNC:[STATe]
value: bool = driver.clock.sync.get_state()
```

No command help available

return
state: No help available

6.3 Connector

class ConnectorCls

Connector 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.connector.clone()
```

Subgroups

6.3.1 RefLo

SCPI Command :

```
CONNector:REFLo:OUTPut
```

class RefLoCls

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

get_output() → EmulSgtRefLoOutput

```
# SCPI: CONNector:REFLo:OUTPut
value: enums.EmulSgtRefLoOutput = driver.connector.refLo.get_output()
```

No command help available

return
output: No help available

set_output(*output: EmulSgtRefLoOutput*) → None

```
# SCPI: CONNector:REFLo:OUTPut
driver.connector.refLo.set_output(output = enums.EmulSgtRefLoOutput.LO)
```

No command help available

param output
No help available

6.3.2 User<UserIx>

RepCap Settings

```
# Range: Nr1 .. Nr48
rc = driver.connector.user.repcap_userIx_get()
driver.connector.user.repcap_userIx_set(repcap.UserIx.Nr1)
```

class UserCls

User commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: UserIx, default value after init: UserIx.Nr1

Cloning the Group

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

Subgroups

6.3.2.1 Clock

class ClockCls

Clock 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.connector.user.clock.clone()
```

Subgroups

6.3.2.1.1 Slope

SCPI Command :

```
CONNector:USER<CH>:CLOCk:SLOPe
```

class SlopeCls

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

get(*userIx=UserIx.Default*) → SlopeType

```
# SCPI: CONNector:USER<CH>:CLOCk:SLOPe
value: enums.SlopeType = driver.connector.user.clock.slope.get(userIx = repcap.
↪UserIx.Default)
```

No command help available

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

return
slope: No help available

set(*slope: SlopeType*, *userIx=UserIx.Default*) → None

```
# SCPI: CONNector:USER<CH>:CLOCk:SLOPe
driver.connector.user.clock.slope.set(slope = enums.SlopeType.NEGative, userIx␣
↪= repcap.UserIx.Default)
```

No command help available

param slope
No help available

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

6.3.2.2 Omode

SCPI Command :

```
CONNector:USER<CH>:OMODe
```

`class OmodeCls`

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

`get`(*userIx=UserIx.Default*) → SgtUserPlug

```
# SCPI: CONNector:USER<CH>:OMODe
value: enums.SgtUserPlug = driver.connector.user.omode.get(userIx = repcap.
↪UserIx.Default)
```

No command help available

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

return
omode: No help available

`set`(*omode: SgtUserPlug*, *userIx=UserIx.Default*) → None

```
# SCPI: CONNector:USER<CH>:OMODe
driver.connector.user.omode.set(omode = enums.SgtUserPlug.CIN, userIx = repcap.
↪UserIx.Default)
```

No command help available

param omode
No help available

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 SMCV100B Service Manual.

param name
test point identifier Test point name, as queried with the command method RsSmcv.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 SMCV100B 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. 19 total commands, 7 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 RsSmcv.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 RsSmcv.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 RsSmcv.Display.Dialog.open method RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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

6.7 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 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

6.8.1 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

6.8.2 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.8.3 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

6.8.3.1 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

6.8.3.1.1 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

6.8.3.1.1.1 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 RsSmcv.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

6.8.3.1.1.2 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 SMCV100B 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

6.8.3.1.1.3 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

6.8.3.1.1.4 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

6.8.3.1.1.5 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

6.8.3.1.1.6 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

6.8.4 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

6.9 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. 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.initiate.clone()
```

Subgroups

6.9.1 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

6.9.1.1 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 RsSmcv. **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 RsSmcv. **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 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

6.11 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 RsSmcv.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

6.11.1 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

6.11.1.1 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 RsSmcv.MassMemory.currentDirectory command.

return
file_count: integer Number of files.

6.11.2 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

6.11.2.1 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 RsSmcv.MassMemory.currentDirectory command.

return
directory_count: integer Number of parent and subdirectories.

6.11.3 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

6.11.3.1 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

6.11.4 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

6.11.4.1 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

6.12 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.

6.13 Output

SCPI Commands :

```
OUTPut<HW>:AMODe
OUTPut<HW>:IMPedance
```

class OutputCls

Output commands group definition. 12 total commands, 5 Subgroups, 2 group commands

get_amode() → PowerAttMode

```
# SCPI: OUTPut<HW>:AMODe
value: enums.PowerAttMode = 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() → InputImpRf

```
# SCPI: OUTPut<HW>:IMPedance
value: enums.InputImpRf = driver.output.get_impedance()
```

Queries the impedance of the RF outputs.

return
impedance: G1K| G50| G10K

set_amode(*amode: PowerAttMode*) → None

```
# SCPI: OUTPut<HW>:AMODe
driver.output.set_amode(amode = enums.PowerAttMode.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

6.13.1 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

6.13.1.1 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.

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.

return
upper: float Unit: dBm

6.13.2 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

6.13.3 Protection

SCPI Commands :

```
OUTPut<HW>:PROTection:CLEar
OUTPut<HW>:PROTection:STATe
OUTPut<HW>:PROTection:TRIPped
```

class ProtectionCls

Protection commands group definition. 3 total commands, 0 Subgroups, 3 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 RsSmcv.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 RsSmcv.Output. State.value.

Same as clear, but waits for the operation to complete before continuing further. Use the RsSmcv.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_state() → bool

```
# SCPI: OUTPut<HW>:PROTection:STATe
value: bool = driver.output.protection.get_state()
```

No command help available

return
state: No help available

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

set_state(*state: bool*) → None

```
# SCPI: OUTPut<HW>:PROTection:STATe
driver.output.protection.set_state(state = False)
```

No command help available

param state
No help available

6.13.4 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

6.13.5 User<UserIx>

RepCap Settings

```
# Range: Nr1 .. Nr48
rc = driver.output.user.repcap_userIx_get()
driver.output.user.repcap_userIx_set(repcap.UserIx.Nr1)
```

class UserCls

User commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: UserIx, default value after init: UserIx.Nr1

Cloning the Group

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

Subgroups

6.13.5.1 Direction

SCPI Command :

```
OUTPut<HW>:USER<CH>:DIRection
```

class DirectionCls

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

get(*userIx=UserIx.Default*) → ConnDirection

```
# SCPI: OUTPut<HW>:USER<CH>:DIRection
value: enums.ConnDirection = driver.output.user.direction.get(userIx = repcap.
↪UserIx.Default)
```

Sets the direction of the signal at the connector that can be input or an output.

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

return
direction: INPut| OUTPut| UNUSed INPut|OUTPut Input signal or output signal UNUSed No signal present at the connector.

set(*direction: ConnDirection*, *userIx=UserIx.Default*) → None

```
# SCPI: OUTPut<HW>:USER<CH>:DIRection
driver.output.user.direction.set(direction = enums.ConnDirection.INPut, userIx=
↪= repcap.UserIx.Default)
```

Sets the direction of the signal at the connector that can be input or an output.

param direction
INPut| OUTPut| UNUSed INPut|OUTPut Input signal or output signal UNUSed No signal present at the connector.

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

6.13.5.2 Signal

SCPI Command :

```
OUTPut<HW>:USER<CH>:SIGNal
```

class SignalCls

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

get(*userIx=UserIx.Default*) → OutpConnGlbSignalb

```
# SCPI: OUTPut<HW>:USER<CH>:SIGNal
value: enums.OutpConnGlbSignalb = driver.output.user.signal.get(userIx = repcap.
↪UserIx.Default)
```

Sets the control signal that is output at the selected connector. To define the connector direction, use the command method RsSmcv.Output.User.Direction.set.

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

return
signal: MARKA1| NONE

set(*signal: OutpConnGlbSignalb*, *userIx=UserIx.Default*) → None

```
# SCPI: OUTPut<HW>:USER<CH>:SIGNal
driver.output.user.signal.set(signal = enums.OutpConnGlbSignalb.MARKA1, userIx=
↪= repcap.UserIx.Default)
```

Sets the control signal that is output at the selected connector. To define the connector direction, use the command method RsSmcv.Output.User.Direction.set.

param signal
MARKA1| NONE

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

6.14 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

6.14.1 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 RsSmcv.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.

6.15 Sconfiguration

SCPI Commands :

```
SCONfiguration:MODE
SCONfiguration:PRESet
```

class SconfigurationCls

Sconfiguration commands group definition. 17 total commands, 5 Subgroups, 2 group commands

get_mode() → EmulSgtBbSystemConfiguration

```
# SCPI: SCONfiguration:MODE
value: enums.EmulSgtBbSystemConfiguration = driver.sconfiguration.get_mode()
```

No command help available

return
configuration: No help available

preset() → None

```
# SCPI: SCONfiguration:PRESet
driver.sconfiguration.preset()
```

No command help available

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

```
# SCPI: SCONfiguration:PRESet
driver.sconfiguration.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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(*configuration: EmulSgtBbSystemConfiguration*) → None

```
# SCPI: SCONfiguration:MODE
driver.sconfiguration.set_mode(configuration = enums.
↪EmulSgtBbSystemConfiguration.AFETracking)
```

No command help available

param configuration
No help available

Cloning the Group

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

Subgroups

6.15.1 Apply

SCPI Command :

```
SCONfiguration:APPLy
```

class ApplyCls

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

set() → None

```
# SCPI: SCONfiguration:APPLy
driver.sconfiguration.apply.set()
```

No command help available

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

```
# SCPI: SCONfiguration:APPLy
driver.sconfiguration.apply.set_with_opc()
```

No command help available

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

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

6.15.2 Baseband

SCPI Command :

```
SCONfiguration:BASeband:SOURce
```

class BasebandCls

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

get_source() → SystConfBbConf

```
# SCPI: SCONfiguration:BASeband:SOURce
value: enums.SystConfBbConf = driver.sconfiguration.baseband.get_source()
```

No command help available

return
sour_config: No help available

set_source(*sour_config: SystConfBbConf*) → None

```
# SCPI: SCONfiguration:BASeband:SOURce
driver.sconfiguration.baseband.set_source(sour_config = enums.SystConfBbConf.
↪COUPled)
```

No command help available

param sour_config
No help available

6.15.3 Diq

class DiqCls

Diq 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.sconfiguration.diq.clone()
```

Subgroups

6.15.3.1 BbMm1

SCPI Command :

```
SCONfiguration:DIQ:BBMM1:CHANnels
```

class BbMm1Cls

BbMm1 commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_channels() → SystConfHsChannels

```
# SCPI: SCONfiguration:DIQ:BBMM1:CHANnels
value: enums.SystConfHsChannels = driver.sconfiguration.diq.bbMm1.get_channels()
```

No command help available

return
dig_iq_hs_bbmm_1_cha: No help available

set_channels(*dig_iq_hs_bbmm_1_cha: SystConfHsChannels*) → None

```
# SCPI: SCONfiguration:DIQ:BBMM1:CHANnels
driver.sconfiguration.diq.bbMm1.set_channels(dig_iq_hs_bbmm_1_cha = enums.
↪SystConfHsChannels.CH0)
```

No command help available

param dig_iq_hs_bbmm_1_cha
No help available

6.15.3.2 BbMm2

SCPI Command :

```
SCONfiguration:DIQ:BBMM2:CHANnels
```

class BbMm2Cls

BbMm2 commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_channels() → SystConfHsChannels

```
# SCPI: SCONfiguration:DIQ:BBMM2:CHANnels
value: enums.SystConfHsChannels = driver.sconfiguration.diq.bbMm2.get_channels()
```

No command help available

return
dig_iq_hs_bbmm_2_cha: No help available

set_channels(*dig_iq_hs_bbmm_2_cha: SystConfHsChannels*) → None

```
# SCPI: SCONfiguration:DIQ:BBMM2:CHANnels
driver.sconfiguration.diq.bbMm2.set_channels(dig_iq_hs_bbmm_2_cha = enums.
↪SystConfHsChannels.CH0)
```

No command help available

param dig_iq_hs_bbmm_2_cha
No help available

6.15.4 MultiInstrument

SCPI Commands :

```
SCONfiguration:MULTiinstrument:MODE
SCONfiguration:MULTiinstrument:STATe
```

class MultiInstrumentCls

MultiInstrument commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_mode() → MultInstMsMode

```
# SCPI: SCONfiguration:MULTiinstrument:MODE
value: enums.MultInstMsMode = driver.sconfiguration.multiInstrument.get_mode()
```

Sets if the instrument works as a primary or as a secondary instrument.

return
ms_mode: PRIMary| SECondary

get_state() → bool

```
# SCPI: SCONfiguration:MULTiinstrument:STATe
value: bool = driver.sconfiguration.multiInstrument.get_state()
```

Activates the selected mode.

return
trigger_state: 1| ON| 0| OFF

set_mode(*ms_mode: MultInstMsMode*) → None

```
# SCPI: SCONfiguration:MULTiinstrument:MODE
driver.sconfiguration.multiInstrument.set_mode(ms_mode = enums.MultInstMsMode.
↪PRIMary)
```

Sets if the instrument works as a primary or as a secondary instrument.

param ms_mode
PRIMary| SECondary

set_state(*trigger_state: bool*) → None

```
# SCPI: SCONfiguration:MULTiinstrument:STATe
driver.sconfiguration.multiInstrument.set_state(trigger_state = False)
```

Activates the selected mode.

param trigger_state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.15.4.1 Connector

class ConnectorCls

Connector 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.sconfiguration.multiInstrument.connector.clone()
```

Subgroups

6.15.4.1.1 Bsin<Channel>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.sconfiguration.multiInstrument.connector.bsin.repcap_channel_get()
driver.sconfiguration.multiInstrument.connector.bsin.repcap_channel_set(repcap.Channel.
↪Nr1)
```

SCPI Command :

```
SCONfiguration:MULTiinstrument:CONNector:BSIN<CH>
```

class BsinCls

Bsin commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

get(*channel=Channel.Default*) → str

```
# SCPI: SCONfiguration:MULTiinstrument:CONNector:BSIN<CH>
value: str = driver.sconfiguration.multiInstrument.connector.bsin.get(channel =␣
↪repcap.Channel.Default)
```

No command help available

param channel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Bsin')

return
connector_name: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.multiInstrument.connector.bsin.clone()
```

6.15.4.1.2 Bsout<ChannelNull>

RepCap Settings

```
# Range: Nr0 .. Nr63
rc = driver.sconfiguration.multiInstrument.connector.bsout.repcap_channelNull_get()
driver.sconfiguration.multiInstrument.connector.bsout.repcap_channelNull_set(repcap.
↪ChannelNull.Nr0)
```

SCPI Command :

```
SCONfiguration:MULTiinstrument:CONNector:BSOut<CH0>
```

class BsoutCls

Bsout commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: ChannelNull, default value after init: ChannelNull.Nr0

get(*channelNull=ChannelNull.Default*) → str

```
# SCPI: SCONfiguration:MULTiinstrument:CONNector:BSOut<CH0>
value: str = driver.sconfiguration.multiInstrument.connector.bsout.
↪get(channelNull = repcap.ChannelNull.Default)
```

No command help available

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bsout')

return
connector_name: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.multiInstrument.connector.bsout.clone()
```

6.15.5 Output

SCPI Command :

```
SCONfiguration:OUTPut:MODE
```

class OutputCls

Output commands group definition. 7 total commands, 1 Subgroups, 1 group commands

get_mode() → SystConfOutpMode

```
# SCPI: SCONfiguration:OUTPut:MODE
value: enums.SystConfOutpMode = driver.sconfiguration.output.get_mode()
```

No command help available

return
mode: No help available

set_mode(*mode: SystConfOutpMode*) → None

```
# SCPI: SCONfiguration:OUTPut:MODE
driver.sconfiguration.output.set_mode(mode = enums.SystConfOutpMode.ALL)
```

No command help available

param mode
No help available

Cloning the Group

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

Subgroups

6.15.5.1 Mapping

class MappingCls

Mapping commands group definition. 6 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.15.5.1.1 Digital

class DigitalCls

Digital 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.sconfiguration.output.mapping.digital.clone()
```

Subgroups

6.15.5.1.1.1 Stream<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.sconfiguration.output.mapping.digital.stream.repcap_stream_get()
driver.sconfiguration.output.mapping.digital.stream.repcap_stream_set(repcap.Stream.Nr1)
```

class StreamCls

Stream commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.output.mapping.digital.stream.clone()
```

Subgroups

6.15.5.1.1.2 State

SCPI Command :

```
SCONfiguration:OUTPut:MAPPing:DIGital:STReam<ST>:STATe
```

class StateCls

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

get(*stream=Stream.Default*) → bool

```
# SCPI: SCONfiguration:OUTPut:MAPPing:DIGital:STReam<ST>:STATe
value: bool = driver.sconfiguration.output.mapping.digital.stream.state.
↪get(stream = repcap.Stream.Default)
```

No command help available

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

return
state: No help available

set(*state: bool*, *stream=Stream.Default*) → None

```
# SCPI: SCONfiguration:OUTPut:MAPPing:DIGital:STReam<ST>:STATe
driver.sconfiguration.output.mapping.digital.stream.state.set(state = False,␣
↪stream = repcap.Stream.Default)
```

No command help available

param state
No help available

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

6.15.5.1.2 HsDigital

class HsDigitalCls

HsDigital 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.sconfiguration.output.mapping.hsDigital.clone()
```

Subgroups

6.15.5.1.2.1 Channel

class ChannelCls

Channel 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.sconfiguration.output.mapping.hsDigital.channel.clone()
```

Subgroups

6.15.5.1.2.2 Stream<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.sconfiguration.output.mapping.hsDigital.channel.stream.repcap_stream_get()
driver.sconfiguration.output.mapping.hsDigital.channel.stream.repcap_stream_set(repcap.
↪Stream.Nr1)
```

class StreamCls

Stream commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.output.mapping.hsDigital.channel.stream.clone()
```

Subgroups

6.15.5.1.2.3 State

SCPI Command :

```
SCONfiguration:OUTPut:MAPPing:HSDigital:CHANnel:STReam<ST>:STATe
```

class StateCls

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

get(*stream=Stream.Default*) → bool

```
# SCPI: SCONfiguration:OUTPut:MAPPing:HSDigital:CHANnel:STReam<ST>:STATe
value: bool = driver.sconfiguration.output.mapping.hsDigital.channel.stream.
↪state.get(stream = repcap.Stream.Default)
```

Maps the I/Q output streams to the output connectors.

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

return
state: 1| ON| 0| OFF

set(*state: bool*, *stream=Stream.Default*) → None

```
# SCPI: SCONfiguration:OUTPut:MAPPing:HSDigital:CHANnel:STReam<ST>:STATe
driver.sconfiguration.output.mapping.hsDigital.channel.stream.state.set(state =␣
↪False, stream = repcap.Stream.Default)
```

Maps the I/Q output streams to the output connectors.

param state
1| ON| 0| OFF

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

6.15.5.1.3 IqOutput

class IqOutputCls

IqOutput 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.sconfiguration.output.mapping.iqOutput.clone()
```

Subgroups

6.15.5.1.3.1 Stream<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.sconfiguration.output.mapping.iqOutput.stream.repcap_stream_get()
driver.sconfiguration.output.mapping.iqOutput.stream.repcap_stream_set(repcap.Stream.Nr1)
```

class StreamCls

Stream commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.output.mapping.iqOutput.stream.clone()
```

Subgroups

6.15.5.1.3.2 State

SCPI Command :

```
SCONfiguration:OUTPut:MAPPing:IQOutput:STReam<ST>:STATe
```

class StateCls

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

get(*stream=Stream.Default*) → bool

```
# SCPI: SCONfiguration:OUTPut:MAPPing:IQOutput:STReam<ST>:STATe
value: bool = driver.sconfiguration.output.mapping.iqOutput.stream.state.
↪get(stream = repcap.Stream.Default)
```

No command help available

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

return
state: No help available

set(*state: bool*, *stream=Stream.Default*) → None

```
# SCPI: SCONfiguration:OUTPut:MAPPing:IQOutput:STReam<ST>:STATe
driver.sconfiguration.output.mapping.iqOutput.stream.state.set(state = False,␣
↪stream = repcap.Stream.Default)
```

No command help available

param state
No help available

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

6.15.5.1.4 Rf<Path>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.sconfiguration.output.mapping.rf.repcap_path_get()
driver.sconfiguration.output.mapping.rf.repcap_path_set(repcap.Path.Nr1)
```

class RfCls

Rf commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Path, default value after init: Path.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.output.mapping.rf.clone()
```

Subgroups

6.15.5.1.4.1 Stream<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.sconfiguration.output.mapping.rf.stream.repcap_stream_get()
driver.sconfiguration.output.mapping.rf.stream.repcap_stream_set(repcap.Stream.Nr1)
```

class StreamCls

Stream commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.output.mapping.rf.stream.clone()
```

Subgroups

6.15.5.1.4.2 State

SCPI Command :

```
SCONfiguration:OUTPut:MAPPing:RF<CH>:STReam<ST>:STATe
```

class StateCls

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

get(*path=Path.Default*, *stream=Stream.Default*) → bool

```
# SCPI: SCONfiguration:OUTPut:MAPPing:RF<CH>:STReam<ST>:STATe
value: bool = driver.sconfiguration.output.mapping.rf.stream.state.get(path =␣
↪repcap.Path.Default, stream = repcap.Stream.Default)
```

Maps the I/Q output streams to the output connectors.

param path
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rf')

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

return
state: 1| ON| 0| OFF

set(*state: bool*, *path=Path.Default*, *stream=Stream.Default*) → None

```
# SCPI: SCONfiguration:OUTPut:MAPPing:RF<CH>:STReam<ST>:STATe
driver.sconfiguration.output.mapping.rf.stream.state.set(state = False, path =␣
↪repcap.Path.Default, stream = repcap.Stream.Default)
```

Maps the I/Q output streams to the output connectors.

param state
1| ON| 0| OFF

param path
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rf')

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

6.15.5.1.5 Stream<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.sconfiguration.output.mapping.stream.repcap_stream_get()
driver.sconfiguration.output.mapping.stream.repcap_stream_set(repcap.Stream.Nr1)
```

class StreamCls

Stream commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sconfiguration.output.mapping.stream.clone()
```

Subgroups

6.15.5.1.5.1 Foffset

SCPI Command :

```
SCONfiguration:OUTPut:MAPPing:STReam<ST>:FOFFset
```

class FoffsetCls

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

get(*stream=Stream.Default*) → float

```
# SCPI: SCONfiguration:OUTPut:MAPPing:STReam<ST>:FOFFset
value: float = driver.sconfiguration.output.mapping.stream.foffset.get(stream =␣
↪repcap.Stream.Default)
```

Sets an absolute frequency offset.

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

return
sm_freq_offset: float Range: depends on the installed options, e.g. -60E6 to +60E6 (base unit)

set(*sm_freq_offset: float*, *stream=Stream.Default*) → None

```
# SCPI: SCONfiguration:OUTPut:MAPPing:STReam<ST>:FOFFset
driver.sconfiguration.output.mapping.stream.foffset.set(sm_freq_offset = 1.0,␣
↪stream = repcap.Stream.Default)
```

Sets an absolute frequency offset.

param sm_freq_offset
float Range: depends on the installed options, e.g. -60E6 to +60E6 (base unit)

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

6.15.5.1.5.2 Poffset

SCPI Command :

```
SCONfiguration:OUTPut:MAPPing:STReam<ST>:POFFset
```

class PoffsetCls

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

get(*stream=Stream.Default*) → float

```
# SCPI: SCONfiguration:OUTPut:MAPPing:STReam<ST>:POFFset
value: float = driver.sconfiguration.output.mapping.stream.poffset.get(stream =␣
↪repcap.Stream.Default)
```

Sets the phase offset of the corresponding stream.

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

return
sm_phas_offset: float Range: -999.99 to 999.99

set(*sm_phas_offset: float*, *stream=Stream.Default*) → None

```
# SCPI: SCONfiguration:OUTPut:MAPPing:STReam<ST>:POFFset
driver.sconfiguration.output.mapping.stream.poffset.set(sm_phas_offset = 1.0,␣
↪stream = repcap.Stream.Default)
```

Sets the phase offset of the corresponding stream.

param sm_phas_offset
float Range: -999.99 to 999.99

param stream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Stream')

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. 25 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. 24 total commands, 14 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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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

set(*status: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:STATus:[DEVice]
driver.sense.power.status.device.set(status = False, channel = repcap.Channel.
↪Default)
```

Queries if a sensor is connected to the instrument.

param status
1| ON| 0| OFF

param channel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

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)
```

No command help available

param channel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

return
sversion: No help available

6.16.1.13 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

6.16.1.14 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 **RsSmcv.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 **RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.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. 1526 total commands, 21 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 RsSmcv.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 Am

SCPI Command :

```
[SOURce<HW>]:AM:SENSitivity
```

class AmCls

Am commands group definition. 4 total commands, 2 Subgroups, 1 group commands

get_sensitivity() → float

```
# SCPI: [SOURce<HW>]:AM:SENSitivity
value: float = driver.source.am.get_sensitivity()
```

No command help available

return
sensitivity: No help available

Cloning the Group

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

Subgroups

6.18.1.1 Bband

SCPI Commands :

```
[SOURce<HW>]:AM:BBANd:SENSitivity
[SOURce<HW>]:AM:BBANd:[STATe]
```

class BbandCls

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

get_sensitivity() → float

```
# SCPI: [SOURce<HW>]:AM:BBANd:SENSitivity
value: float = driver.source.am.bband.get_sensitivity()
```

No command help available

return
sensitivity: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:AM:BBANd:[STATe]
value: bool = driver.source.am.bband.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:AM:BBANd:[STATe]
driver.source.am.bband.set_state(state = False)
```

No command help available

param state
No help available

6.18.1.2 External

SCPI Command :

```
[SOURce<HW>]:AM:EXTernal:COUPling
```

class ExternalCls

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

get_coupling() → AcDc

```
# SCPI: [SOURce<HW>]:AM:EXTernal:COUPling
value: enums.AcDc = driver.source.am.external.get_coupling()
```

No command help available

return
coupling: No help available

set_coupling(*coupling: AcDc*) → None

```
# SCPI: [SOURce<HW>]:AM:EXTernal:COUPling
driver.source.am.external.set_coupling(coupling = enums.AcDc.AC)
```

No command help available

param coupling
No help available

6.18.2 Awgn

SCPI Commands :

```
[SOURce<HW>]:AWGN:BRATe
[SOURce<HW>]:AWGN:CNRatio
[SOURce<HW>]:AWGN:ENRatio
[SOURce<HW>]:AWGN:MODE
[SOURce<HW>]:AWGN:STATe
```

class AwgnCls

Awgn commands group definition. 21 total commands, 5 Subgroups, 5 group commands

get_brate() → float

```
# SCPI: [SOURce<HW>]:AWGN:BRATe
value: float = driver.source.awgn.get_brate()
```

Sets the bit rate used for calculation of bit energy to noise power ratio. Valid units are bps, kbps and mabps as well as b/s, kb/s and mab/s.

return
brate: float Range: 400 to depends on the installed options

get_cn_ratio() → float

```
# SCPI: [SOURce<HW>]:AWGN:CNRatio
value: float = driver.source.awgn.get_cn_ratio()
```

Sets the carrier/interferer ratio.

return
cn_ratio: float Range: -50 to 45

get_en_ratio() → float

```
# SCPI: [SOURce<HW>]:AWGN:ENRatio
value: float = driver.source.awgn.get_en_ratio()
```

Sets the ratio of bit energy to noise power density.

return
en_ratio: float Range: -50 to depends on the installed options, Unit: dB

get_mode() → NoisAwgnMode

```
# SCPI: [SOURce<HW>]:AWGN:MODE
value: enums.NoisAwgnMode = driver.source.awgn.get_mode()
```

Determines how the interfering signal is generated.

return
mode: ONLY| ADD| CW ADD The AWGN noise signal is added to the baseband signal. ONLY The pure AWGN noise signal is modulated to the carrier. The connection to the baseband is interrupted CW The sine interfering signal is added to the baseband signal.

get_state() → bool

```
# SCPI: [SOURce<HW>]:AWGN:STATe
value: bool = driver.source.awgn.get_state()
```

Activates or deactivates the AWGN generator.

return
state: 1| ON| 0| OFF

set_brate(*brate: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:BRATe
driver.source.awgn.set_brate(brate = 1.0)
```

Sets the bit rate used for calculation of bit energy to noise power ratio. Valid units are bps, kbps and mabps as well as b/s, kb/s and mab/s.

param brate
float Range: 400 to depends on the installed options

set_cn_ratio(*cn_ratio: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:CNRatio
driver.source.awgn.set_cn_ratio(cn_ratio = 1.0)
```

Sets the carrier/interferer ratio.

param cn_ratio
float Range: -50 to 45

set_en_ratio(*en_ratio: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:ENRatio
driver.source.awgn.set_en_ratio(en_ratio = 1.0)
```

Sets the ratio of bit energy to noise power density.

param en_ratio
float Range: -50 to depends on the installed options, Unit: dB

set_mode(*mode: NoisAwgnMode*) → None

```
# SCPI: [SOURce<HW>]:AWGN:MODE
driver.source.awgn.set_mode(mode = enums.NoisAwgnMode.ADD)
```

Determines how the interfering signal is generated.

param mode
ONLY| ADD| CW ADD The AWGN noise signal is added to the baseband signal. ONLY The pure AWGN noise signal is modulated to the carrier. The connection to the baseband is interrupted CW The sine interfering signal is added to the baseband signal.

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:AWGN:STATe
driver.source.awgn.set_state(state = False)
```

Activates or deactivates the AWGN generator.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.18.2.1 Bandwidth

SCPI Commands :

```
[SOURce<HW>]:AWGN:BWIDth:NOISe
[SOURce<HW>]:AWGN:BWIDth:RATio
[SOURce<HW>]:AWGN:BWIDth
```

class BandwidthCls

Bandwidth commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_noise() → float

```
# SCPI: [SOURce<HW>]:AWGN:BWIDth:NOISe
value: float = driver.source.awgn.bandwidth.get_noise()
```

Queries the real noise bandwidth.

return
noise: float Range: 0 to 200E6

get_ratio() → float

```
# SCPI: [SOURce<HW>]:AWGN:BWIDth:RATio
value: float = driver.source.awgn.bandwidth.get_ratio()
```

Sets the ratio of minimum real noise bandwidth to system bandwidth, see also 'Signal and noise parameters'.

return
ratio: float Range: 1 to Max

get_value() → float

```
# SCPI: [SOURce<HW>]:AWGN:BWIDth
value: float = driver.source.awgn.bandwidth.get_value()
```

Sets the system bandwidth.

return
bwidth: float Range: 1000 to 80E6

set_ratio(*ratio: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:BWIDth:RATio
driver.source.awgn.bandwidth.set_ratio(ratio = 1.0)
```

Sets the ratio of minimum real noise bandwidth to system bandwidth, see also 'Signal and noise parameters'.

param ratio
float Range: 1 to Max

set_value(*bwidth: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:BWIDth
driver.source.awgn.bandwidth.set_value(bwidth = 1.0)
```

Sets the system bandwidth.

param bwidth
float Range: 1000 to 80E6

Cloning the Group

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

Subgroups

6.18.2.1.1 Coupling

SCPI Command :

```
[SOURce<HW>]:AWGN:BWIDth:COUPling:[STATe]
```

class CouplingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:AWGN:BWIDth:COUPling:[STATe]
value: bool = driver.source.awgn.bandwidth.coupling.get_state()
```

Activates bandwidth coupling.If activated, the digital broadcast baseband signal bandwidth couples to the AWGN system bandwidth.

return
awgn_bw_coup_state: 1| ON| 0| OFF

set_state(*awgn_bw_coup_state: bool*) → None

```
# SCPI: [SOURce<HW>]:AWGN:BWIDth:COUPling:[STATe]
driver.source.awgn.bandwidth.coupling.set_state(awgn_bw_coup_state = False)
```

Activates bandwidth coupling.If activated, the digital broadcast baseband signal bandwidth couples to the AWGN system bandwidth.

param awgn_bw_coup_state
1| ON| 0| OFF

6.18.2.2 Cmode

SCPI Command :

```
[SOURce<HW>]:AWGN:CMODe:[STATe]
```

class CmodeCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:AWGN:CMODe:[STATe]
value: bool = driver.source.awgn.cmode.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:AWGN:CMODe:[STATe]
driver.source.awgn.cmode.set_state(state = False)
```

No command help available

param state
No help available

6.18.2.3 Disp

SCPI Commands :

```
[SOURce<HW>]:AWGN:DISP:MODE
[SOURce<HW>]:AWGN:DISP:ORESults
```

class DispCls

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

get_mode() → NoisAwgnDispMode

```
# SCPI: [SOURce<HW>]:AWGN:DISP:MODE
value: enums.NoisAwgnDispMode = driver.source.awgn.disp.get_mode()
```

Sets the output to that the AWGN settings are related.

return
mode: RFA| IQOUT1

get_oresults() → AnalogDigital

```
# SCPI: [SOURce<HW>]:AWGN:DISP:ORESults
value: enums.AnalogDigital = driver.source.awgn.disp.get_oresults()
```

No command help available

return
oresults: No help available

set_oresults(*oresults: AnalogDigital*) → None

```
# SCPI: [SOURce<HW>]:AWGN:DISP:ORESults
driver.source.awgn.disp.set_oresults(oresults = enums.AnalogDigital.ANALog)
```

No command help available

param oresults
No help available

6.18.2.4 Frequency

SCPI Commands :

```
[SOURce<HW>]:AWGN:FREQuency:RESult
[SOURce<HW>]:AWGN:FREQuency:TARGet
```

class FrequencyCls

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

get_result() → float

```
# SCPI: [SOURce<HW>]:AWGN:FREQuency:RESult
value: float = driver.source.awgn.frequency.get_result()
```

Queries the actual frequency of the sine wave.

return
result: float Range: -40E6 to 40E6

get_target() → float

```
# SCPI: [SOURce<HW>]:AWGN:FREQuency:TARGet
value: float = driver.source.awgn.frequency.get_target()
```

Sets the desired frequency of the sine wave.

return
target: float Range: -40E6 to 40E6

set_target(*target: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:FREQuency:TARGet
driver.source.awgn.frequency.set_target(target = 1.0)
```

Sets the desired frequency of the sine wave.

param target
float Range: -40E6 to 40E6

6.18.2.5 Power

SCPI Commands :

```
[SOURce<HW>]:AWGN:POWer:CARRier
[SOURce<HW>]:AWGN:POWer:MODE
[SOURce<HW>]:AWGN:POWer:RMODe
```

class PowerCls

Power commands group definition. 7 total commands, 2 Subgroups, 3 group commands

get_carrier() → float

```
# SCPI: [SOURce<HW>]:AWGN:POWer:CARRier
value: float = driver.source.awgn.power.get_carrier()
```

Sets the carrier power.

return
carrier: float

get_mode() → NoisAwgnPowMode

```
# SCPI: [SOURce<HW>]:AWGN:POWer:MODE
value: enums.NoisAwgnPowMode = driver.source.awgn.power.get_mode()
```

Selects the mode for setting the noise level.

return
mode: CN| SN | EN

get_rmode() → NoisAwgnPowRefMode

```
# SCPI: [SOURce<HW>]:AWGN:POWer:RMODe
value: enums.NoisAwgnPowRefMode = driver.source.awgn.power.get_rmode()
```

Determines whether the carrier or the noise level is kept constant when the C/N value or Eb/N0 value is changed.

return
rmode: CARRier| NOISe

set_carrier(*carrier: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:POWer:CARRier
driver.source.awgn.power.set_carrier(carrier = 1.0)
```

Sets the carrier power.

param carrier
float

set_mode(*mode: NoisAwgnPowMode*) → None

```
# SCPI: [SOURce<HW>]:AWGN:POWer:MODE
driver.source.awgn.power.set_mode(mode = enums.NoisAwgnPowMode.CN)
```

Selects the mode for setting the noise level.

param mode
CN| SN | EN

set_rmode(*rmode: NoisAwgnPowRefMode*) → None

```
# SCPI: [SOURce<HW>]:AWGN:POWer:RMODe
driver.source.awgn.power.set_rmode(rmode = enums.NoisAwgnPowRefMode.CARRier)
```

Determines whether the carrier or the noise level is kept constant when the C/N value or Eb/N0 value is changed.

param rmode
CARRier| NOISe

Cloning the Group

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

Subgroups

6.18.2.5.1 Noise

SCPI Commands :

```
[SOURce<HW>]:AWGN:POWer:NOISe:TOTal
[SOURce<HW>]:AWGN:POWer:NOISe
```

class NoiseCls

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

get_total() → float

```
# SCPI: [SOURce<HW>]:AWGN:POWer:NOISe:TOTal
value: float = driver.source.awgn.power.noise.get_total()
```

Queries the noise level in the total bandwidth.

return
total: float Range: -145 to 20

get_value() → float

```
# SCPI: [SOURce<HW>]:AWGN:POWer:NOISe
value: float = driver.source.awgn.power.noise.get_value()
```

Sets the power of the noise signal in the system respectively total bandwidth.

return
noise: float

set_value(*noise: float*) → None

```
# SCPI: [SOURce<HW>]:AWGN:POWer:NOISe
driver.source.awgn.power.noise.set_value(noise = 1.0)
```

Sets the power of the noise signal in the system respectively total bandwidth.

param noise
float

6.18.2.5.2 Sum

SCPI Commands :

```
[SOURce<HW>]:AWGN:POWer:SUM:PEP
[SOURce<HW>]:AWGN:POWer:SUM
```

class SumCls

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

get_pep() → float

```
# SCPI: [SOURce<HW>]:AWGN:POWer:SUM:PEP
value: float = driver.source.awgn.power.sum.get_pep()
```

Queries the peak envelope power of the overall signal comprised of noise signal plus useful signal.

return
pep: float Range: -145 to 20

get_value() → float

```
# SCPI: [SOURce<HW>]:AWGN:POWer:SUM
value: float = driver.source.awgn.power.sum.get_value()
```

Queries the overall power of the noise/interferer signal plus useful signal

return
sum: float Range: -145 to 20

6.18.3 Bb

SCPI Commands :

```
[SOURce<HW>]:BB:CFACtor
[SOURce]:BB:CONFiguration
[SOURce<HW>]:BB:FOFFset
[SOURce<HW>]:BB:IQGain
[SOURce<HW>]:BB:PGAin
[SOURce<HW>]:BB:POFFset
[SOURce<HW>]:BB:ROUTe
```

class BbCls

Bb commands group definition. 1134 total commands, 26 Subgroups, 7 group commands

get_cfactor() → float

```
# SCPI: [SOURce<HW>]:BB:CFACtor
value: float = driver.source.bb.get_cfactor()
```

Queries the crest factor of the baseband signal.

return
cfactor: float Range: 0 to 100, Unit: dB

get_configuration() → BbConfig

```
# SCPI: [SOURce]:BB:CONFiguration
value: enums.BbConfig = driver.source.bb.get_configuration()
```

No command help available

return
configuration: No help available

get_foffset() → float

```
# SCPI: [SOURce<HW>]:BB:FOFFset
value: float = driver.source.bb.get_foffset()
```

Sets a frequency offset for the internal/external baseband signal. The offset affects the generated baseband signal.

return
foffset: float Range: depends on the installed options , Unit: Hz

get_iq_gain() → IqGain

```
# SCPI: [SOURce<HW>]:BB:IQGain
value: enums.IqGain = driver.source.bb.get_iq_gain()
```

No command help available

return
ipartq_gain: No help available

get_pgain() → float

```
# SCPI: [SOURce<HW>]:BB:PGAin
value: float = driver.source.bb.get_pgain()
```

No command help available

return
pgain: No help available

get_poffset() → float

```
# SCPI: [SOURce<HW>]:BB:POFFset
value: float = driver.source.bb.get_poffset()
```

Sets the relative phase offset for the selected baseband signal.

return
poffset: float Range: 0 to 359.9, Unit: DEG

get_route() → PathUniCodBbin

```
# SCPI: [SOURce<HW>]:BB:ROUTe
value: enums.PathUniCodBbin = driver.source.bb.get_route()
```

Selects the signal route for the internal/external baseband signal.

return
route: A

set_configuration(*configuration: BbConfig*) → None

```
# SCPI: [SOURce]:BB:CONFiguration
driver.source.bb.set_configuration(configuration = enums.BbConfig.NORMal)
```

No command help available

param configuration
No help available

set_foffset(*foffset: float*) → None

```
# SCPI: [SOURce<HW>]:BB:FOFFset
driver.source.bb.set_foffset(foffset = 1.0)
```

Sets a frequency offset for the internal/external baseband signal. The offset affects the generated baseband signal.

param foffset
float Range: depends on the installed options , Unit: Hz

set_iq_gain(*ipartq_gain: IqGain*) → None

```
# SCPI: [SOURce<HW>]:BB:IQGain
driver.source.bb.set_iq_gain(ipartq_gain = enums.IqGain.DB0)
```

No command help available

param ipartq_gain
No help available

set_pgain(*pgain: float*) → None

```
# SCPI: [SOURce<HW>]:BB:PGAin
driver.source.bb.set_pgain(pgain = 1.0)
```

No command help available

param pgain
No help available

set_poffset(*poffset: float*) → None

```
# SCPI: [SOURce<HW>]:BB:POFFset
driver.source.bb.set_poffset(poffset = 1.0)
```

Sets the relative phase offset for the selected baseband signal.

param poffset
float Range: 0 to 359.9, Unit: DEG

set_route(*route: PathUniCodBbin*) → None

```
# SCPI: [SOURce<HW>]:BB:ROUTe
driver.source.bb.set_route(route = enums.PathUniCodBbin.A)
```

Selects the signal route for the internal/external baseband signal.

param route
A

Cloning the Group

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

Subgroups

6.18.3.1 A3Tsc

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:BSID
[SOURce<HW>]:BB:A3TSc:IPPacket
[SOURce<HW>]:BB:A3TSc:LLS
[SOURce<HW>]:BB:A3TSc:NETWorkmode
[SOURce<HW>]:BB:A3TSc:NRF
[SOURce<HW>]:BB:A3TSc:PAPR
[SOURce<HW>]:BB:A3TSc:PAYLoad
[SOURce<HW>]:BB:A3TSc:PID
[SOURce<HW>]:BB:A3TSc:PIDTestpack
[SOURce<HW>]:BB:A3TSc:PRESet
[SOURce<HW>]:BB:A3TSc:SOURce
[SOURce<HW>]:BB:A3TSc:STATe
[SOURce<HW>]:BB:A3TSc:TIME
[SOURce<HW>]:BB:A3TSc:TSPacket
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.clone()
```

Subgroups

6.18.3.1.1 Channel

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:CHANnel:[BANDwidth]
```

6.18.3.1.2 Delay

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:DELay:DEViation
[SOURce<HW>]:BB:A3TSc:DELay:DISPatch
[SOURce<HW>]:BB:A3TSc:DELay:DYNamic
[SOURce<HW>]:BB:A3TSc:DELay:MAXImum
[SOURce<HW>]:BB:A3TSc:DELay:NETWork
[SOURce<HW>]:BB:A3TSc:DELay:PROCess
[SOURce<HW>]:BB:A3TSc:DELay:SFNMode
[SOURce<HW>]:BB:A3TSc:DELay:STATic
[SOURce<HW>]:BB:A3TSc:DELay:TOTal
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.delay.clone()
```

Subgroups

6.18.3.1.2.1 Mute

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:DELay:MUTE:[BOOTstrap]
```

6.18.3.1.2.2 Tsp

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:DELay:TSP:LTT
[SOURce<HW>]:BB:A3TSc:DELay:TSP:LTU
[SOURce<HW>]:BB:A3TSc:DELay:TSP:TOET
[SOURce<HW>]:BB:A3TSc:DELay:TSP:UTO
```

6.18.3.1.3 Frame

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:FRAMe:EXFinal
[SOURce<HW>]:BB:A3TSc:FRAMe:EXSYmbol
[SOURce<HW>]:BB:A3TSc:FRAMe:LENGth
[SOURce<HW>]:BB:A3TSc:FRAMe:MODE
[SOURce<HW>]:BB:A3TSc:FRAMe:NSUBframes
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.frame.clone()
```

Subgroups

6.18.3.1.3.1 Additional

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:FRAMe:ADDitional:[SAMPles]
```

6.18.3.1.3.2 Time

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:FRAMe:TIME:[OFFSet]
```

6.18.3.1.4 Info

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.info.clone()
```

Subgroups

6.18.3.1.4.1 Bootstrap

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:BANDwidth
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:DURation
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:EAS
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:MAJor
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:MINor
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.info.bootstrap.clone()
```

Subgroups

6.18.3.1.4.2 Basic

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:BASic:FECType
```

6.18.3.1.4.3 Bsr

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:BSR:COEFficient
```

6.18.3.1.4.4 Fft

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:FFT:MODE
```

6.18.3.1.4.5 Guard

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:GUARd:INTerval
```

6.18.3.1.4.6 Pilot

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:PILot:DX
```

6.18.3.1.4.7 Preamble

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:PREamble:[STRucture]
```

6.18.3.1.4.8 Time

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:BOOTstrap:TIME:NEXT
```

6.18.3.1.4.9 Frame

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INFO:FRAMe:DURation
```

6.18.3.1.4.10 Lpy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.info.lpy.clone()
```

Subgroups

6.18.3.1.4.11 Basic

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:INFO:L:BASic:BYTes
[SOURce<HW>]:BB:A3TSc:INFO:L:BASic:CELLs
```

6.18.3.1.4.12 Detail

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:INFO:L:DETail:BYTes
[SOURce<HW>]:BB:A3TSc:INFO:L:DETail:CELLs
```

6.18.3.1.5 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:INPut:CCHeck
[SOURce<HW>]:BB:A3TSc:INPut:NPLP
[SOURce<HW>]:BB:A3TSc:INPut:PROTocol
[SOURce<HW>]:BB:A3TSc:INPut:STATus
[SOURce<HW>]:BB:A3TSc:INPut:TYPE
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.inputPy.clone()
```

Subgroups

6.18.3.1.5.1 Destination

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.inputPy.destination.clone()
```

Subgroups

6.18.3.1.5.2 Ip

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:INPut:DESTination:IP:ADDRess
[SOURce<HW>]:BB:A3TSc:INPut:DESTination:IP:PORT
```

6.18.3.1.5.3 Stl

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INPut:STL:INTerface
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.inputPy.stl.clone()
```

Subgroups

6.18.3.1.5.4 ResetLog

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:INPut:STL:RESetlog
```

6.18.3.1.6 Lpy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.lpy.clone()
```

Subgroups

6.18.3.1.6.1 Basic

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:L:BASic:FECType
[SOURce<HW>]:BB:A3TSc:L:BASic:VERSion
```

6.18.3.1.6.2 Carrier

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:L:CARRier:MODE
```

6.18.3.1.6.3 Detail

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:L:DETail:FECType
[SOURce<HW>]:BB:A3TSc:L:DETail:VERSion
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.lpy.detail.clone()
```

Subgroups

6.18.3.1.6.4 Additional

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:L:DETail:ADDitional:[PARity]
```

6.18.3.1.6.5 Npreamble

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:L:NPReamble:[SYMBols]
```

6.18.3.1.6.6 Pilot

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:L:PILot:DX
```

6.18.3.1.7 Miso

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:MISo:IDX
[SOURce<HW>]:BB:A3TSc:MISo:NTX
```

6.18.3.1.8 Plp<PhysicalLayerPipe>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.a3Tsc.plp.repcap_physicalLayerPipe_get()
driver.source.bb.a3Tsc.plp.repcap_physicalLayerPipe_set(repcap.PhysicalLayerPipe.Nr1)
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.clone()
```

Subgroups

6.18.3.1.8.1 AlpType

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:ALPType
```

6.18.3.1.8.2 BbfCounter

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:BBFCounter
```

6.18.3.1.8.3 BbfPadding

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:BBFPadding
```

6.18.3.1.8.4 Constel

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:CONStel
```

6.18.3.1.8.5 FecType

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:FECType
```

6.18.3.1.8.6 Id

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:ID
```

6.18.3.1.8.7 InputPy

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP:INPut:TESTsignal
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.inputPy.clone()
```

Subgroups

6.18.3.1.8.8 DataRate

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:[INPut]:DATarate
```

6.18.3.1.8.9 Layer

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.layer.clone()
```

Subgroups

6.18.3.1.8.10 Layer

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:LAYer:LAYer
```

6.18.3.1.8.11 Level

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:LAYer:LEVel
```

6.18.3.1.8.12 Lls

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:LLS
```

6.18.3.1.8.13 PacketLength

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:PACKetlength
```

6.18.3.1.8.14 Rate

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:RATE
```

6.18.3.1.8.15 Scrambler

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:SCRambler
```

6.18.3.1.8.16 Size

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:SIZE
```

6.18.3.1.8.17 Til

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.til.clone()
```

Subgroups

6.18.3.1.8.18 Blocks

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:BLOCks
```

6.18.3.1.8.19 Cil

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:CIL
```

6.18.3.1.8.20 Depth

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:DEPTh
```

6.18.3.1.8.21 Extended

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:EXTended
```

6.18.3.1.8.22 Inter

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:INTer
```

6.18.3.1.8.23 MaxBlocks

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:MAXBlocks
```

6.18.3.1.8.24 NtiBlocks

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:NTIBlocks
```

6.18.3.1.8.25 Til

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TIL:TIL
```

6.18.3.1.8.26 TypePy

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.typePy.clone()
```

Subgroups

6.18.3.1.8.27 NsubSlices

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TYPE:NSUBslices
```

6.18.3.1.8.28 Subslice

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.typePy.subslice.clone()
```

Subgroups

6.18.3.1.8.29 Interval

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TYPE:SUBSlice:[INTerval]
```

6.18.3.1.8.30 TypePy

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:TYPE:TYPE
```

6.18.3.1.8.31 Useful

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.useful.clone()
```

Subgroups

6.18.3.1.8.32 Rate

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:USEFul:[RATe]
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.plp.useful.rate.clone()
```

Subgroups

6.18.3.1.8.33 Max

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PLP<CH>:USEFul:[RATe]:MAX
```

6.18.3.1.9 Prbs

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:PRBS:[SEQuence]
```

6.18.3.1.10 Return

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:RETurn:[CHANnel]
```

6.18.3.1.11 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:SETTing:CATalog
[SOURce<HW>]:BB:A3TSc:SETTing:DELete
[SOURce<HW>]:BB:A3TSc:SETTing:LOAD
[SOURce<HW>]:BB:A3TSc:SETTing:STORe
```

6.18.3.1.12 Special

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.special.clone()
```

Subgroups

6.18.3.1.12.1 Alp

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SPECial:ALP:LMT
```

6.18.3.1.12.2 Bootstrap

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:SPECial:BOOTstrap:EAS
[SOURce<HW>]:BB:A3TSc:SPECial:BOOTstrap:MINor
```

6.18.3.1.12.3 Settings

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SPECial:SETTings:[STATe]
```

6.18.3.1.12.4 Stl

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:SPECial:STL:PREamble
[SOURce<HW>]:BB:A3TSc:SPECial:STL:TMP
```

6.18.3.1.13 Subframe<Subframe>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.source.bb.a3Tsc.subframe.repcap_subframe_get()
driver.source.bb.a3Tsc.subframe.repcap_subframe_set(repcap.Subframe.Nr1)
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.clone()
```

Subgroups

6.18.3.1.13.1 Carrier

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.carrier.clone()
```

Subgroups

6.18.3.1.13.2 Mode

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:CARRier:MODE
```

6.18.3.1.13.3 Duration

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:DURation
```

6.18.3.1.13.4 Fft

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.fft.clone()
```

Subgroups

6.18.3.1.13.5 Mode

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:FFT:MODE
```

6.18.3.1.13.6 Fil

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:FIL
```

6.18.3.1.13.7 Guard

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.guard.clone()
```

Subgroups

6.18.3.1.13.8 Interval

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:GUARd:INTerval
```

6.18.3.1.13.9 Mimo

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:MIMO
```

6.18.3.1.13.10 Miso

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:MISO
```

6.18.3.1.13.11 Ndata

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:NDATa
```

6.18.3.1.13.12 Pilot

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.pilot.clone()
```

Subgroups

6.18.3.1.13.13 Boost

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:PILot:BOOSt
```

6.18.3.1.13.14 Siso

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:PILot:SISO
```

6.18.3.1.13.15 Plp

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.plp.clone()
```

Subgroups

6.18.3.1.13.16 NidPlp

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:PLP:NIDPlp
```

6.18.3.1.13.17 Nplp

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:PLP:NPLP
```

6.18.3.1.13.18 Sbs

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.sbs.clone()
```

Subgroups

6.18.3.1.13.19 First

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:SBS:FIRSt
```

6.18.3.1.13.20 Last

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:SBS:LAST
```

6.18.3.1.13.21 Null

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:SBS:NULL
```

6.18.3.1.13.22 Used

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.a3Tsc.subframe.used.clone()
```

Subgroups

6.18.3.1.13.23 Bandwidth

SCPI Command :

```
[SOURce<HW>]:BB:A3TSc:SUBFrame<CH>:USED:[BANDwidth]
```

6.18.3.1.14 Txid

SCPI Commands :

```
[SOURce<HW>]:BB:A3TSc:TXId:ADDRess
[SOURce<HW>]:BB:A3TSc:TXId:LEVel
[SOURce<HW>]:BB:A3TSc:TXId:MODE
```

6.18.3.2 Arbitrary

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:PRESet
[SOURce<HW>]:BB:ARBitrary:STATe
```

class ArbitraryCls

Arbitrary commands group definition. 141 total commands, 9 Subgroups, 2 group commands

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:STATe
value: bool = driver.source.bb.arbitrary.get_state()
```

Enables the ARB generator. A waveform must be selected before the ARB generator is activated.

return
state: 1| ON| 0| OFF

preset() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:PRESet
driver.source.bb.arbitrary.preset()
```

Sets all ARB generator parameters to their default values.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:PRESet
driver.source.bb.arbitrary.preset_with_opc()
```

Sets all ARB generator parameters to their default values.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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:ARBitrary:STATe
driver.source.bb.arbitrary.set_state(state = False)
```

Enables the ARB generator. A waveform must be selected before the ARB generator is activated.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.18.3.2.1 Cfr

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:CFR:ALGorithm
[SOURce<HW>]:BB:ARBitrary:CFR:CPBandwidth
[SOURce<HW>]:BB:ARBitrary:CFR:CSPacing
[SOURce<HW>]:BB:ARBitrary:CFR:DCFDelta
[SOURce<HW>]:BB:ARBitrary:CFR:FILTer
[SOURce<HW>]:BB:ARBitrary:CFR:FORDer
[SOURce<HW>]:BB:ARBitrary:CFR:ITERations
[SOURce<HW>]:BB:ARBitrary:CFR:OCFactor
[SOURce<HW>]:BB:ARBitrary:CFR:PFReq
[SOURce<HW>]:BB:ARBitrary:CFR:RCFactor
[SOURce<HW>]:BB:ARBitrary:CFR:SBANdwidth
[SOURce<HW>]:BB:ARBitrary:CFR:SFReq
```

(continues on next page)

(continued from previous page)

```
[SOURce<HW>]:BB:ARBitrary:CFR:TBANdwidth
[SOURce<HW>]:BB:ARBitrary:CFR:[STATe]
```

class CfrCls

Cfr commands group definition. 17 total commands, 3 Subgroups, 14 group commands

get_algorithm() → CfrAlgo

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:ALGorithm
value: enums.CfrAlgo = driver.source.bb.arbitrary.cfr.get_algorithm()
```

Defines the algorithm for crest factor reduction.

return
arb_cfr_algorithm: CLFiltering| PCANcellation CLFiltering Clipping and filtering algorithm. This algorithm performs a hard clipping of the baseband signal. It is followed by a low pass filtering of the result in an iterative manner until the target crest factor is reached. You can define the settings of the filter that is used for the calculation. PCANcellation Peak cancelation algorithm. This algorithm subtracts Blackman windowed sinc pulses from the signal wherever the amplitude is above a defined threshold.

get_cp_bandwidth() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:CPBandwidth
value: float = driver.source.bb.arbitrary.cfr.get_cp_bandwidth()
```

Sets the cancellation pulse bandwidth for peak cancellation CFR algorithm.

return
arb_cfr_canc_pul_bw: float Range: 0 to 250E6

get_cspacing() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:CSPacing
value: float = driver.source.bb.arbitrary.cfr.get_cspacing()
```

Sets the channel spacing, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to SIMPle.

return
arb_cfr_chan_spac: float Range: 0 to depends on the sample rate of the loaded file

get_dcfdelta() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:DCFDelta
value: float = driver.source.bb.arbitrary.cfr.get_dcfdelta()
```

Sets the value difference by which you want to change your crest factor.

return
arb_cfr_dcf_delta: float Range: -20 to 0

get_filter_py() → CfrFiltMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:FILTer
value: enums.CfrFiltMode = driver.source.bb.arbitrary.cfr.get_filter_py()
```

Selects which filter mode is used for the filtering.

return
arb_cfr_filter_mod: SIMPle| ENHanced

get_forder() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:FORDer
value: int = driver.source.bb.arbitrary.cfr.get_forder()
```

Sets the maximum filter order, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to ENHanced.

return
arb_cfr_max_file_order: integer Range: 0 to 300

get_iterations() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:ITERations
value: int = driver.source.bb.arbitrary.cfr.get_iterations()
```

Sets the number of iterations that are used for calculating the resulting crest factor. The iteration process is stopped when the desired crest factor delta is achieved by 0.1 dB.

return
arb_cfr_max_iter: integer Range: 1 to 10

get_oc_factor() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:OCFactor
value: float = driver.source.bb.arbitrary.cfr.get_oc_factor()
```

Queries the original crest factor of the waveform after the calculation of the resulting crest factor is completed. The original crest factor is calculated as an average over the whole waveform, including any idle periods that might be present in TDD waveforms.

return
arb_cfro_crest_factor: float Range: 1 to 100

get_pfreq() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:PFReq
value: float = driver.source.bb.arbitrary.cfr.get_pfreq()
```

Sets the passband frequency, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to ENHanced. Frequency components lower than the passband frequency are passed through unfiltered.

return
arb_cfr_pass_band_freq: float Range: 0 to depends on the sample rate of the loaded file

get_rc_factor() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:RCFactor
value: float = driver.source.bb.arbitrary.cfr.get_rc_factor()
```

Queries the resulting crest factor of the waveform after the calculations are completed. The resulting crest factor is calculated as an average over the whole waveform, including any idle periods that might be present in TDD waveforms.

return
arb_cfr_res_crest_factor: float Range: 1 to 100

get_sbandwidth() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:SBANdwidth
value: float = driver.source.bb.arbitrary.cfr.get_sbandwidth()
```

Sets the signal bandwidth, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to SIMPle. The value of the signal bandwidth should not be higher than the channel spacing ([:SOURce<hw>]:BB:ARBitrary:CFR:CSPacing) .

return
arb_cfr_signal_bw: float Range: 0 to depends on the sample rate of the loaded file

get_sfreq() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:SFReq
value: float = driver.source.bb.arbitrary.cfr.get_sfreq()
```

Sets the stopband frequency of the filter, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to ENHanced. Frequency components higher than the stopband frequency are filtered out by the lowpass filter.

return
arb_cfr_stop_band_freq: float Range: 0 to depends on the sample rate of the loaded file

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:[STATe]
value: bool = driver.source.bb.arbitrary.cfr.get_state()
```

Enables the crest factor reduction calculation.

return
arb_cfr_state: 1| ON| 0| OFF

get_tbandwidth() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:TBANdwidth
value: float = driver.source.bb.arbitrary.cfr.get_tbandwidth()
```

Sets the transition bandwidth of the cancellation pulse for peak cancellation CFR algorithm.

return
dda_rb_cfr_tran_bw: float Range: 0 to 250E6

set_algorithm(*arb_cfr_algorithm: CfrAlgo*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:ALGorithm
driver.source.bb.arbitrary.cfr.set_algorithm(arb_cfr_algorithm = enums.CfrAlgo.
↪CLFiltering)
```

Defines the algorithm for crest factor reduction.

param arb_cfr_algorithm
CLFiltering| PCANcellation CLFiltering Clipping and filtering algorithm. This algorithm performs a hard clipping of the baseband signal. It is followed by a low pass filtering of the result in an iterative manner until the target crest factor is reached. You can define the settings of the filter that is used for the calculation. PCANcellation Peak cancelation algorithm. This algorithm subtracts Blackman windowed sinc pulses from the signal wherever the amplitude is above a defined threshold.

set_cp_bandwidth(*arb_cfr_canc_pul_bw: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:CPBandwidth
driver.source.bb.arbitrary.cfr.set_cp_bandwidth(arb_cfr_canc_pul_bw = 1.0)
```

Sets the cancellation pulse bandwidth for peak cancellation CFR algorithm.

param arb_cfr_canc_pul_bw
float Range: 0 to 250E6

set_cspacing(*arb_cfr_chan_spac: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:CSPacing
driver.source.bb.arbitrary.cfr.set_cspacing(arb_cfr_chan_spac = 1.0)
```

Sets the channel spacing, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to SIMPle.

param arb_cfr_chan_spac
float Range: 0 to depends on the sample rate of the loaded file

set_dcfdelta(*arb_cfr_dcf_delta: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:DCFDelta
driver.source.bb.arbitrary.cfr.set_dcfdelta(arb_cfr_dcf_delta = 1.0)
```

Sets the value difference by which you want to change your crest factor.

param arb_cfr_dcf_delta
float Range: -20 to 0

set_filter_py(*arb_cfr_filter_mod: CfrFiltMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:FILTer
driver.source.bb.arbitrary.cfr.set_filter_py(arb_cfr_filter_mod = enums.
↪CfrFiltMode.ENHanced)
```

Selects which filter mode is used for the filtering.

param arb_cfr_filter_mod
SIMPle| ENHanced

set_forder(*arb_cfr_max_file_order: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:FORDer
driver.source.bb.arbitrary.cfr.set_forder(arb_cfr_max_file_order = 1)
```

Sets the maximum filter order, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to ENHanced.

param arb_cfr_max_file_order
integer Range: 0 to 300

set_iterations(*arb_cfr_max_iter: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:ITERations
driver.source.bb.arbitrary.cfr.set_iterations(arb_cfr_max_iter = 1)
```

Sets the number of iterations that are used for calculating the resulting crest factor. The iteration process is stopped when the desired crest factor delta is achieved by 0.1 dB.

param arb_cfr_max_iter
integer Range: 1 to 10

set_pfreq(*arb_cfr_pass_band_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:PFReq
driver.source.bb.arbitrary.cfr.set_pfreq(arb_cfr_pass_band_freq = 1.0)
```

Sets the passband frequency, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to ENHanced. Frequency components lower than the passband frequency are passed through unfiltered.

param arb_cfr_pass_band_freq
float Range: 0 to depends on the sample rate of the loaded file

set_sbandwidth(*arb_cfr_signal_bw: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:SBANdwidth
driver.source.bb.arbitrary.cfr.set_sbandwidth(arb_cfr_signal_bw = 1.0)
```

Sets the signal bandwidth, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to SIMPle. The value of the signal bandwidth should not be higher than the channel spacing ([:SOURce<hw>]:BB:ARBitrary:CFR:CSPacing) .

param arb_cfr_signal_bw
float Range: 0 to depends on the sample rate of the loaded file

set_sfreq(*arb_cfr_stop_band_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:SFReq
driver.source.bb.arbitrary.cfr.set_sfreq(arb_cfr_stop_band_freq = 1.0)
```

Sets the stopband frequency of the filter, if [:SOURce<hw>]:BB:ARBitrary:CFR:FILTer is set to ENHanced. Frequency components higher than the stopband frequency are filtered out by the lowpass filter.

param arb_cfr_stop_band_freq
float Range: 0 to depends on the sample rate of the loaded file

set_state(*arb_cfr_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:[STATe]
driver.source.bb.arbitrary.cfr.set_state(arb_cfr_state = False)
```

Enables the crest factor reduction calculation.

param arb_cfr_state
1| ON| 0| OFF

set_tbandwidth(*dda_rb_cfr_tran_bw: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:TBANdwidth
driver.source.bb.arbitrary.cfr.set_tbandwidth(dda_rb_cfr_tran_bw = 1.0)
```

Sets the transition bandwidth of the cancellation pulse for peak cancellation CFR algorithm.

param dda_rb_cfr_tran_bw
float Range: 0 to 250E6

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.cfr.clone()
```

Subgroups

6.18.3.2.1.1 CfWaveform

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:CFR:CFWaveform:[STATe]
```

class CfWaveformCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:CFWaveform:[STATe]
value: bool = driver.source.bb.arbitrary.cfr.cfWaveform.get_state()
```

No command help available

return
arb_cfr_cfw_state: No help available

set_state(*arb_cfr_cfw_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:CFWaveform:[STATe]
driver.source.bb.arbitrary.cfr.cfWaveform.set_state(arb_cfr_cfw_state = False)
```

No command help available

param arb_cfr_cfw_state
No help available

6.18.3.2.1.2 Measure

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:CFR:MEASure:[STATe]
```

class MeasureCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:MEASure:[STATe]
value: bool = driver.source.bb.arbitrary.cfr.measure.get_state()
```

Queries the state of the crest factor reduction calculation.

return
measure_state: 1| ON| 0| OFF ON: the original and resulting crest factors are already calculated.

6.18.3.2.1.3 Waveform

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:CFR:WAVeform:CREate
```

class WaveformCls

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

set_create(*create_wv_file: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CFR:WAVeform:CREate
driver.source.bb.arbitrary.cfr.waveform.set_create(create_wv_file = 'abc')
```

With enabled signal generation, triggers the instrument to save the current settings in a waveform file. Waveform files can be further processed. The filename and the directory it is saved in are user-definable; the predefined file extension for waveform files is *.wv.

param create_wv_file
string

6.18.3.2.2 Clock

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:CLOCk:MODE
[SOURce<HW>]:BB:ARBitrary:CLOCk:MULTiplier
[SOURce<HW>]:BB:ARBitrary:CLOCk:SOURce
[SOURce<HW>]:BB:ARBitrary:CLOCk
```

class ClockCls

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

get_mode() → ClockModeUnits

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:MODE
value: enums.ClockModeUnits = driver.source.bb.arbitrary.clock.get_mode()
```

No command help available

return
mode: No help available

get_multiplier() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:MULTiplier
value: int = driver.source.bb.arbitrary.clock.get_multiplier()
```

No command help available

return
multiplier: No help available

get_source() → ClocSourBb

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:SOURce
value: enums.ClocSourBb = driver.source.bb.arbitrary.clock.get_source()


    INTRO_CMD_HELP: Selects the clock source:

    - INTernal: Internal clock reference

    :return: source: INTernal
```

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk
value: float = driver.source.bb.arbitrary.clock.get_value()
```

Sets the clock frequency. If you load a waveform, the clock rate is determined as defined with the waveform tag {CLOCK: frequency}. This command subsequently changes the clock rate; see data sheet for value range.

return
clock: float Range: depends on the installed options , Unit: Hz E.g. 400 Hz to 300 MHz

set_mode(*mode: ClockModeUnits*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:MODE
driver.source.bb.arbitrary.clock.set_mode(mode = enums.ClockModeUnits.MSAMple)
```

No command help available

param mode
No help available

set_multiplier(*multiplier: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:MULTiplier
driver.source.bb.arbitrary.clock.set_multiplier(multiplier = 1)
```

No command help available

param multiplier
No help available

set_source(*source: ClocSourBb*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:SOURce
driver.source.bb.arbitrary.clock.set_source(source = enums.ClocSourBb.AINTernal)


    INTRO_CMD_HELP: Selects the clock source:
```

(continues on next page)

(continued from previous page)

```
    - INTernal: Internal clock reference

    :param source: INTernal
```

set_value(*clock: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk
driver.source.bb.arbitrary.clock.set_value(clock = 1.0)
```

Sets the clock frequency. If you load a waveform, the clock rate is determined as defined with the waveform tag {CLOCK: frequency}. This command subsequently changes the clock rate; see data sheet for value range.

param clock
float Range: depends on the installed options , Unit: Hz E.g. 400 Hz to 300 MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.clock.clone()
```

Subgroups

6.18.3.2.2.1 Synchronization

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:CLOCk:SYNChronization:MODE
```

class SynchronizationCls

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

get_mode() → ClocSyncModeSgt

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:SYNChronization:MODE
value: enums.ClocSyncModeSgt = driver.source.bb.arbitrary.clock.synchronization.
↪get_mode()
```

No command help available

return
mode: No help available

set_mode(*mode: ClocSyncModeSgt*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:SYNChronization:MODE
driver.source.bb.arbitrary.clock.synchronization.set_mode(mode = enums.
↪ClocSyncModeSgt.DIIN)
```

No command help available

param mode
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.clock.synchronization.clone()
```

Subgroups

6.18.3.2.2.2 Execute

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:CLOCk:SYNChronization:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:SYNChronization:EXECute
driver.source.bb.arbitrary.clock.synchronization.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:CLOCk:SYNChronization:EXECute
driver.source.bb.arbitrary.clock.synchronization.execute.set_with_opc()
```

No command help available

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

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

6.18.3.2.3 Mcarrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.arbitrary.mcarrier.repcap_carrier_get()
driver.source.bb.arbitrary.mcarrier.repcap_carrier_set(repcap.Carrier.Nr1)
```

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CLOCk
[SOURce<HW>]:BB:ARBitrary:MCARrier:OFILe
[SOURce<HW>]:BB:ARBitrary:MCARrier:PRESet
[SOURce<HW>]:BB:ARBitrary:MCARrier:SAMPles
```

class McarrierCls

Mcarrier commands group definition. 38 total commands, 9 Subgroups, 4 group commands Repeated Capability: Carrier, default value after init: Carrier.Nr1

get_clock() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLOCk
value: float = driver.source.bb.arbitrary.mcarrier.get_clock()
```

Queries the resulting sample rate at which the multi-carrier waveform is output by the arbitrary waveform generator. The output clock rate depends on the number of carriers, carrier spacing, and input sample rate of the leftmost or rightmost carriers.

return
clock: float Range: 400 to Max

get_ofile() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:OFILe
value: str = driver.source.bb.arbitrary.mcarrier.get_ofile()
```

Defines the output file name for the multi-carrier waveform (file extension *.wv) . This file name is required to calculate the waveform with the commands [:SOURce<hw>]:BB:ARBitrary:MCARrier:CLOad or [:SOURce<hw>]:BB:ARBitrary:MCARrier:CREate.

return
ofile: string

get_samples() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:SAMPles
value: int = driver.source.bb.arbitrary.mcarrier.get_samples()
```

Queries the resulting file size.

return
samples: integer Range: 0 to INT_MAX, Unit: samples

preset() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:PRESet
driver.source.bb.arbitrary.mcarrier.preset()
```

Sets all the multi-carrier parameters to their default values.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:PRESet
driver.source.bb.arbitrary.mcarrier.preset_with_opc()
```

Sets all the multi-carrier parameters to their default values.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_ofile(*ofile: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:OFILe
driver.source.bb.arbitrary.mcarrier.set_ofile(ofile = 'abc')
```

Defines the output file name for the multi-carrier waveform (file extension *.wv) . This file name is required to calculate the waveform with the commands [:SOURce<hw>]:BB:ARBitrary:MCARrier:CLOad or [:SOURce<hw>]:BB:ARBitrary:MCARrier:CREate.

param ofile
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.mcarrier.clone()
```

Subgroups

6.18.3.2.3.1 Carrier

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:COUNt
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:MODE
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:SPACing
```

class CarrierCls

Carrier commands group definition. 10 total commands, 7 Subgroups, 3 group commands

get_count() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:COUNt
value: int = driver.source.bb.arbitrary.mcarrier.carrier.get_count()
```

Sets the number of carriers in the ARB multi-carrier waveform.

return
count: integer Range: 1 to 512

get_mode() → ArbMultCarrSpacMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:MODE
value: enums.ArbMultCarrSpacMode = driver.source.bb.arbitrary.mcarrier.carrier.
↪get_mode()
```

The command sets the carrier frequency mode.

return
mode: EQUidistant| ARBitrary

get_spacing() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:SPACing
value: float = driver.source.bb.arbitrary.mcarrier.carrier.get_spacing()
```

Sets the frequency spacing between adjacent carriers of the multi-carrier waveform (see 'Defining the carrier frequency') .

return
spacing: float Range: 0.0 to depends on the installed options, e.g. 120E6, Unit: Hz

set_count(*count: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:COUNt
driver.source.bb.arbitrary.mcarrier.carrier.set_count(count = 1)
```

Sets the number of carriers in the ARB multi-carrier waveform.

param count
integer Range: 1 to 512

set_mode(*mode: ArbMultCarrSpacMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:MODE
driver.source.bb.arbitrary.mcarrier.carrier.set_mode(mode = enums.
↪ArbMultCarrSpacMode.ARBitrary)
```

The command sets the carrier frequency mode.

param mode
EQUidistant| ARBitrary

set_spacing(*spacing: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier:SPACing
driver.source.bb.arbitrary.mcarrier.carrier.set_spacing(spacing = 1.0)
```

Sets the frequency spacing between adjacent carriers of the multi-carrier waveform (see 'Defining the carrier frequency') .

param spacing
float Range: 0.0 to depends on the installed options, e.g. 120E6, Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.mcarrier.carrier.clone()
```

Subgroups

6.18.3.2.3.2 Conflict

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:CONFlict
```

class ConflictCls

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

get(*carrier=Carrier.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:CONFlict
value: bool = driver.source.bb.arbitrary.mcarrier.carrier.conflict.get(carrier␣
↪= repcap.Carrier.Default)
```

Queries carrier conflicts. A conflict arises when the carriers overlap.

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

return
conflict: 1| ON| 0| OFF 0 No conflict

6.18.3.2.3.3 Delay

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:DELay
```

class DelayCls

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

get(*carrier=Carrier.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:DELay
value: float = driver.source.bb.arbitrary.mcarrier.carrier.delay.get(carrier =␣
↪repcap.Carrier.Default)
```

Sets the start delay of the selected carrier.

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

return
delay: float Range: 0 to 1, Unit: s

set(*delay: float*, *carrier=Carrier.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:DELay
driver.source.bb.arbitrary.mcarrier.carrier.delay.set(delay = 1.0, carrier =␣
↪repcap.Carrier.Default)
```

Sets the start delay of the selected carrier.

param delay
float Range: 0 to 1, Unit: s

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

6.18.3.2.3.4 File

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:FILE
```

class FileCls

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

get(*carrier=Carrier.Default*) → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:FILE
value: str = driver.source.bb.arbitrary.mcarrier.carrier.file.get(carrier =␣
↪repcap.Carrier.Default)
```

Selects the file with I/Q data to be modulated onto the selected carrier.

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

return
file: file name

set(*file: str*, *carrier=Carrier.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:FILE
driver.source.bb.arbitrary.mcarrier.carrier.file.set(file = 'abc', carrier =␣
↪repcap.Carrier.Default)
```

Selects the file with I/Q data to be modulated onto the selected carrier.

param file
file name

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

6.18.3.2.3.5 Frequency

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:FREQuency
```

class FrequencyCls

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

get(*carrier=Carrier.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:FREQuency
value: int = driver.source.bb.arbitrary.mcarrier.carrier.frequency.get(carrier␣
↪= repcap.Carrier.Default)
```

Sets or indicates the carrier frequency, depending on the selected carrier frequency mode.

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

return
frequency: integer Range: depends on the installed options

set(*frequency: int*, *carrier=Carrier.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:FREQuency
driver.source.bb.arbitrary.mcarrier.carrier.frequency.set(frequency = 1,␣
↪carrier = repcap.Carrier.Default)
```

Sets or indicates the carrier frequency, depending on the selected carrier frequency mode.

param frequency
integer Range: depends on the installed options

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

6.18.3.2.3.6 Phase

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:PHASe
```

class PhaseCls

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

get(*carrier=Carrier.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:PHASe
value: float = driver.source.bb.arbitrary.mcarrier.carrier.phase.get(carrier =␣
↪repcap.Carrier.Default)
```

Sets the start phase of the selected carrier.

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

return
phase: float Range: 0 to 359.99, Unit: DEG

set(*phase: float*, *carrier=Carrier.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:PHASe
driver.source.bb.arbitrary.mcarrier.carrier.phase.set(phase = 1.0, carrier =␣
↪repcap.Carrier.Default)
```

Sets the start phase of the selected carrier.

param phase
float Range: 0 to 359.99, Unit: DEG

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

6.18.3.2.3.7 Power

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:POWer
```

class PowerCls

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

get(*carrier=Carrier.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:POWer
value: float = driver.source.bb.arbitrary.mcarrier.carrier.power.get(carrier =␣
↪repcap.Carrier.Default)
```

Sets the gain of the selected carrier.

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

return
power: float Range: -80 to 0, Unit: dB

set(*power: float*, *carrier=Carrier.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:POWer
driver.source.bb.arbitrary.mcarrier.carrier.power.set(power = 1.0, carrier =␣
↪repcap.Carrier.Default)
```

Sets the gain of the selected carrier.

param power
float Range: -80 to 0, Unit: dB

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

6.18.3.2.3.8 State

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:STATe
```

class StateCls

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

get(*carrier=Carrier.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:STATe
value: bool = driver.source.bb.arbitrary.mcarrier.carrier.state.get(carrier =␣
↪repcap.Carrier.Default)
```

Enables/diasbled the selected carrier.

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

return
state: 1| ON| 0| OFF

set(*state: bool*, *carrier=Carrier.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CARRier<CH>:STATe
driver.source.bb.arbitrary.mcarrier.carrier.state.set(state = False, carrier =␣
↪repcap.Carrier.Default)
```

Enables/diasbled the selected carrier.

param state
1| ON| 0| OFF

param carrier
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mcarrier')

6.18.3.2.3.9 Cfactor

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CFACtor:MODE
```

class CfactorCls

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

get_mode() → ArbMultCarrCresMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CFACtor:MODE
value: enums.ArbMultCarrCresMode = driver.source.bb.arbitrary.mcarrier.cfactor.
↪get_mode()
```

Sets the mode for optimizing the crest factor by calculating the carrier phases.

return
mode: OFF| MIN| MAX

set_mode(*mode: ArbMultCarrCresMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CFACtor:MODE
driver.source.bb.arbitrary.mcarrier.cfactor.set_mode(mode = enums.
↪ArbMultCarrCresMode.MAX)
```

Sets the mode for optimizing the crest factor by calculating the carrier phases.

param mode
OFF| MIN| MAX

6.18.3.2.3.10 Clipping

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:CFACtor
[SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:CUToff
[SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:[STATe]
```

class ClippingCls

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

get_cfactor() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:CFACtor
value: float = driver.source.bb.arbitrary.mcarrier.clipping.get_cfactor()
```

Sets the value of the desired crest factor, if baseband clipping is enabled. A target crest factor above the crest factor of the unclipped multicarrier signal has no effect.

return
cfactor: float Range: -50 to 50, Unit: dB

get_cutoff() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:CUToff
value: float = driver.source.bb.arbitrary.mcarrier.clipping.get_cutoff()
```

Sets the cutoff frequency of the final low pass filter, if baseband clipping is enabled.

return
cutoff: float Range: 0 to 250E6

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:[STATe]
value: bool = driver.source.bb.arbitrary.mcarrier.clipping.get_state()
```

Switches baseband clipping on and off.

return
state: 1| ON| 0| OFF

set_cfactor(*cfactor: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:CFACtor
driver.source.bb.arbitrary.mcarrier.clipping.set_cfactor(cfactor = 1.0)
```

Sets the value of the desired crest factor, if baseband clipping is enabled. A target crest factor above the crest factor of the unclipped multicarrier signal has no effect.

param cfactor
float Range: -50 to 50, Unit: dB

set_cutoff(*cutoff: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:CUToff
driver.source.bb.arbitrary.mcarrier.clipping.set_cutoff(cutoff = 1.0)
```

Sets the cutoff frequency of the final low pass filter, if baseband clipping is enabled.

param cutoff
float Range: 0 to 250E6

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLIPping:[STATe]
driver.source.bb.arbitrary.mcarrier.clipping.set_state(state = False)
```

Switches baseband clipping on and off.

param state
1| ON| 0| OFF

6.18.3.2.3.11 Cload

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CLOad
```

class CloadCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLOad
driver.source.bb.arbitrary.mcarrier.cload.set()
```

Creates a multi-carrier waveform using the current entries of the carrier table and activates the ARB generator. Use the command [:SOURce<hw>]:BB:ARBitrary:MCARrier:OFILe to define the multi-carrier waveform file name. The file extension is *. wv.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CLOad
driver.source.bb.arbitrary.mcarrier.cload.set_with_opc()
```

Creates a multi-carrier waveform using the current entries of the carrier table and activates the ARB generator. Use the command [:SOURce<hw>]:BB:ARBitrary:MCARrier:OFILe to define the multi-carrier waveform file name. The file extension is *. wv.

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

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

6.18.3.2.3.12 Create

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:CREate
```

class CreateCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CREate
driver.source.bb.arbitrary.mcarrier.create.set()
```

Creates a multi-carrier waveform using the current settings of the carrier table. Use the command [:SOURce<hw>]:BB:ARBitrary:MCARrier:OFILe to define the multi-carrier waveform file name. The file extension is *.wv.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:CREate
driver.source.bb.arbitrary.mcarrier.create.set_with_opc()
```

Creates a multi-carrier waveform using the current settings of the carrier table. Use the command [:SOURce<hw>]:BB:ARBitrary:MCARrier:OFILe to define the multi-carrier waveform file name. The file extension is *.wv.

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

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

6.18.3.2.3.13 Edit

class EditCls

Edit commands group definition. 11 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.mcarrier.edit.clone()
```

Subgroups

6.18.3.2.3.14 Carrier

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:FILE
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STARt
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STATe
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STOP
```

class CarrierCls

Carrier commands group definition. 11 total commands, 4 Subgroups, 4 group commands

get_file() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:FILE
value: str = driver.source.bb.arbitrary.mcarrier.edit.carrier.get_file()
```

Selects the input file. The data of the file are modulated onto the carriers in the defined carrier range.

return
file: string

get_start() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STARt
value: int = driver.source.bb.arbitrary.mcarrier.edit.carrier.get_start()
```

Selects the last carrier in the carrier range to which the settings shall apply.

return
start: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STATe
value: bool = driver.source.bb.arbitrary.mcarrier.edit.carrier.get_state()
```

Switches all the carriers in the selected carrier range on or off.

return
state: 1| ON| 0| OFF

get_stop() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STOP
value: int = driver.source.bb.arbitrary.mcarrier.edit.carrier.get_stop()
```

Selects the last carrier in the carrier range to which the settings shall apply.

return
stop: integer Range: 0 to 511

set_file(*file: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:FILE
driver.source.bb.arbitrary.mcarrier.edit.carrier.set_file(file = 'abc')
```

Selects the input file. The data of the file are modulated onto the carriers in the defined carrier range.

param file
string

set_start(*start: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STARt
driver.source.bb.arbitrary.mcarrier.edit.carrier.set_start(start = 1)
```

Selects the last carrier in the carrier range to which the settings shall apply.

param start
integer Range: 0 to 511

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STATe
driver.source.bb.arbitrary.mcarrier.edit.carrier.set_state(state = False)
```

Switches all the carriers in the selected carrier range on or off.

param state
1| ON| 0| OFF

set_stop(*stop: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:STOP
driver.source.bb.arbitrary.mcarrier.edit.carrier.set_stop(stop = 1)
```

Selects the last carrier in the carrier range to which the settings shall apply.

param stop
integer Range: 0 to 511

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.mcarrier.edit.carrier.clone()
```

Subgroups

6.18.3.2.3.15 Delay

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:DELay:STEP
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:DELay:[STARt]
```

class DelayCls

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

get_start() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:DELay:[STARt]
value: float = driver.source.bb.arbitrary.mcarrier.edit.carrier.delay.get_
↪start()
```

Sets the start delay for the individual carriers in the defined carrier range.

return
start: float Range: 0 to 1, Unit: s

get_step() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:DELay:STEP
value: float = driver.source.bb.arbitrary.mcarrier.edit.carrier.delay.get_step()
```

Sets the step width by which the start delays of the carriers in the defined carrier range is incremented.

return
step: float Range: -1 to 1, Unit: s

set_start(*start: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:DELay:[STARt]
driver.source.bb.arbitrary.mcarrier.edit.carrier.delay.set_start(start = 1.0)
```

Sets the start delay for the individual carriers in the defined carrier range.

param start
float Range: 0 to 1, Unit: s

set_step(*step: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:DELay:STEP
driver.source.bb.arbitrary.mcarrier.edit.carrier.delay.set_step(step = 1.0)
```

Sets the step width by which the start delays of the carriers in the defined carrier range is incremented.

param step
float Range: -1 to 1, Unit: s

6.18.3.2.3.16 Execute

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:EXECute
driver.source.bb.arbitrary.mcarrier.edit.carrier.execute.set()
```

Adopts the settings for the selected carrier range.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:EXECute
driver.source.bb.arbitrary.mcarrier.edit.carrier.execute.set_with_opc()
```

Adopts the settings for the selected carrier range.

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

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

6.18.3.2.3.17 Phase

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:PHASe:STEP
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:PHASe:[STARt]
```

class PhaseCls

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

get_start() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:PHASe:[STARt]
value: float = driver.source.bb.arbitrary.mcarrier.edit.carrier.phase.get_
↪start()
```

Sets the start phase for the individual carriers in the defined carrier range.

return
start: float Range: 0 to 359.99, Unit: DEG

get_step() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:PHASe:STEP
value: float = driver.source.bb.arbitrary.mcarrier.edit.carrier.phase.get_step()
```

Sets the step width by which the start phases of the carriers in the defined carrier range is incremented.

return
step: float Range: -359.99 to 359.99, Unit: DEG

set_start(*start: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:PHASe:[STARt]
driver.source.bb.arbitrary.mcarrier.edit.carrier.phase.set_start(start = 1.0)
```

Sets the start phase for the individual carriers in the defined carrier range.

param start
float Range: 0 to 359.99, Unit: DEG

set_step(*step: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:PHASe:STEP
driver.source.bb.arbitrary.mcarrier.edit.carrier.phase.set_step(step = 1.0)
```

Sets the step width by which the start phases of the carriers in the defined carrier range is incremented.

param step
float Range: -359.99 to 359.99, Unit: DEG

6.18.3.2.3.18 Power

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:POWer:STEP
[SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:POWer:[STARt]
```

class PowerCls

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

get_start() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:POWer:[STARt]
value: float = driver.source.bb.arbitrary.mcarrier.edit.carrier.power.get_
↪start()
```

Sets the power for the individual carriers in the defined carrier range.

return
start: float Range: -80 to 0, Unit: dB

get_step() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:POWer:STEP
value: float = driver.source.bb.arbitrary.mcarrier.edit.carrier.power.get_step()
```

Sets the step width by which the starting power of the carriers in the defined carrier range is incremented.

return
step: float Range: -80 to 80, Unit: dB

set_start(*start: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:POWer:[STARt]
driver.source.bb.arbitrary.mcarrier.edit.carrier.power.set_start(start = 1.0)
```

Sets the power for the individual carriers in the defined carrier range.

param start
float Range: -80 to 0, Unit: dB

set_step(*step: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:EDIT:CARRier:POWer:STEP
driver.source.bb.arbitrary.mcarrier.edit.carrier.power.set_step(step = 1.0)
```

Sets the step width by which the starting power of the carriers in the defined carrier range is incremented.

param step
float Range: -80 to 80, Unit: dB

6.18.3.2.3.19 Power

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:POWer:REFerence
```

class PowerCls

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

get_reference() → ArbMultCarrLevRef

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:POWer:REFerence
value: enums.ArbMultCarrLevRef = driver.source.bb.arbitrary.mcarrier.power.get_
↪reference()
```

Defines the way the individual carriers in a composed multi carrier signal are leveled.

return
reference: RMS| PEAK

set_reference(*reference: ArbMultCarrLevRef*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:POWer:REFerence
driver.source.bb.arbitrary.mcarrier.power.set_reference(reference = enums.
↪ArbMultCarrLevRef.PEAK)
```

Defines the way the individual carriers in a composed multi carrier signal are leveled.

param reference
RMS| PEAK

6.18.3.2.3.20 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:CATalog
[SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:LOAD
```

class SettingCls

Setting commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:CATalog
value: List[str] = driver.source.bb.arbitrary.mcarrier.setting.get_catalog()
```

Queries the available settings files in the specified default directory. Only files with the file extension *. arb_multcarr are listed.

return
catalog: string

load(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:LOAD
driver.source.bb.arbitrary.mcarrier.setting.load(filename = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.arb_multcarr. 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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.mcarrier.setting.clone()
```

Subgroups

6.18.3.2.3.21 Store

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:STORe:FAST
[SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:STORe
```

class StoreCls

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

get_fast() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:STORe:FAST
value: bool = driver.source.bb.arbitrary.mcarrier.setting.store.get_fast()
```

No command help available

return
fast: No help available

set_fast(*fast: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:STORe:FAST
driver.source.bb.arbitrary.mcarrier.setting.store.set_fast(fast = False)
```

No command help available

param fast
No help available

set_value(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:SETTing:STORe
driver.source.bb.arbitrary.mcarrier.setting.store.set_value(filename = 'abc')
```

Stores the current settings into the selected file; the file extension (*.arb_multcarr) 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
string Filename or complete file path

6.18.3.2.3.22 Time

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:MCARrier:TIME:MODE
[SOURce<HW>]:BB:ARBitrary:MCARrier:TIME
```

class TimeCls

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

get_mode() → ArbMultCarrSigDurMod

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:TIME:MODE
value: enums.ArbMultCarrSigDurMod = driver.source.bb.arbitrary.mcarrier.time.
↪get_mode()
```

Selects the mode for calculating the resulting signal period of the multi-carrier waveform. The resulting period is always calculated for all carriers in the carrier table irrespective of their state (ON/OFF) .

return
mode: USER| LONG| SHORt| LCM

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:TIME
value: float = driver.source.bb.arbitrary.mcarrier.time.get_value()
```

Sets the user-defined signal period.

return
time: float Range: 0 to 1E9, Unit: s

set_mode(*mode: ArbMultCarrSigDurMod*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:TIME:MODE
driver.source.bb.arbitrary.mcarrier.time.set_mode(mode = enums.
↪ArbMultCarrSigDurMod.LCM)
```

Selects the mode for calculating the resulting signal period of the multi-carrier waveform. The resulting period is always calculated for all carriers in the carrier table irrespective of their state (ON/OFF) .

param mode
USER| LONG| SHORt| LCM

set_value(*time: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:MCARrier:TIME
driver.source.bb.arbitrary.mcarrier.time.set_value(time = 1.0)
```

Sets the user-defined signal period.

param time
float Range: 0 to 1E9, Unit: s

6.18.3.2.4 Pramp

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:PRAMp:[STATe]
```

class PrampCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:PRAMp:[STATe]
value: bool = driver.source.bb.arbitrary.pramp.get_state()
```

No command help available

return
arb_pram_state: No help available

set_state(*arb_pram_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:PRAMp:[STATe]
driver.source.bb.arbitrary.pramp.set_state(arb_pram_state = False)
```

No command help available

param arb_pram_state
No help available

6.18.3.2.5 Signal

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:SIGNal:TYPE
```

class SignalCls

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

get_type_py() → ArbSignType

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:SIGNal:TYPE
value: enums.ArbSignType = driver.source.bb.arbitrary.signal.get_type_py()
```

Selects the type of test signal.

return
arb_signal_type: SINE| RECT| CIQ

set_type_py(*arb_signal_type: ArbSignType*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:SIGNal:TYPE
driver.source.bb.arbitrary.signal.set_type_py(arb_signal_type = enums.
↪ArbSignType.AWGN)
```

Selects the type of test signal.

param arb_signal_type
SINE| RECT| CIQ

6.18.3.2.6 Trigger

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:PTIMe
[SOURce<HW>]:BB:ARBitrary:TRIGger:RMODe
[SOURce<HW>]:BB:ARBitrary:TRIGger:SLENgth
[SOURce<HW>]:BB:ARBitrary:TRIGger:SLUNit
[SOURce<HW>]:BB:ARBitrary:TRIGger:SMODe
[SOURce<HW>]:BB:ARBitrary:TRIGger:SOURce
[SOURce<HW>]:BB:ARBitrary:[TRIGger]:SEQuence
```

class TriggerCls

Trigger commands group definition. 29 total commands, 6 Subgroups, 7 group commands

get_ptime() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:PTIMe
value: str = driver.source.bb.arbitrary.trigger.get_ptime()
```

Queries the internal processing time. The processing time is the elapsed time between the input of the external trigger event and the output of the baseband signal.

return
arb_trig_proc_time: string

get_rmode() → TrigRunMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:RMODe
value: enums.TrigRunMode = driver.source.bb.arbitrary.trigger.get_rmode()
```

Queries the status of waveform output.

return
rmode: STOP| RUN

get_sequence() → DmTrigMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:[TRIGger]:SEQuence
value: enums.DmTrigMode = driver.source.bb.arbitrary.trigger.get_sequence()
```

The command selects the trigger mode.

return
sequence: AUTO| RETRigger| AAUTo| ARETrigger| SINGle

get_sl_unit() → UnitSlB

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SLUNit
value: enums.UnitSlB = driver.source.bb.arbitrary.trigger.get_sl_unit()
```

Defines the unit for the entry of the length of the signal sequence to be output in the Single trigger mode.

return
sl_unit: SEQuence| SAMPle

get_slength() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SLENgth
value: int = driver.source.bb.arbitrary.trigger.get_slength()
```

Defines the length of the signal sequence that is output in the SINGle trigger mode.

return
slength: integer Maximun value dependents on the selected units [:SOURcehw]:BB:ARBitrary:TRIGger:SLUNit as follows: SAMPle: Max = 232-1 SEQuence: Max = 1000 Range: 1 to dynamic

get_smode() → ArbTrigSegmModeNoEhop

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SMODe
value: enums.ArbTrigSegmModeNoEhop = driver.source.bb.arbitrary.trigger.get_
↪smode()
```

Selects the extended trigger mode for multi segment waveforms.

return
smode: SAME| NEXT| SEQuencer| NSEam NSEam = Next Segment Seamless

get_source() → TrigSour

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SOURce
value: enums.TrigSour = driver.source.bb.arbitrary.trigger.get_source()


    INTRO_CMD_HELP: Selects the trigger signal source and determines the way␣
↪the triggering is executed. Provided are:

    - Internal triggering by a command (INTernal)

    :return: source: INTernal| EXTernal
```

set_sequence(*sequence: DmTrigMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:[TRIGger]:SEQuence
driver.source.bb.arbitrary.trigger.set_sequence(sequence = enums.DmTrigMode.
↪AAUTo)
```

The command selects the trigger mode.

param sequence
AUTO| RETRigger| AAUTo| ARETrigger| SINGle

set_sl_unit(*sl_unit: UnitSlB*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SLUNit
driver.source.bb.arbitrary.trigger.set_sl_unit(sl_unit = enums.UnitSlB.SAMPle)
```

Defines the unit for the entry of the length of the signal sequence to be output in the Single trigger mode.

param sl_unit
SEQuence| SAMPle

set_slength(*slength: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SLENgth
driver.source.bb.arbitrary.trigger.set_slength(slength = 1)
```

Defines the length of the signal sequence that is output in the SINGle trigger mode.

param slength
integer Maximun value dependents on the selected units [:SOURcehw]:BB:ARBitrary:TRIGger:SLUNit as follows: SAMPle: Max = 232-1 SEQuence: Max = 1000 Range: 1 to dynamic

set_smode(*smode: ArbTrigSegmModeNoEhop*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SMODe
driver.source.bb.arbitrary.trigger.set_smode(smode = enums.
↪ArbTrigSegmModeNoEhop.NEXT)
```

Selects the extended trigger mode for multi segment waveforms.

param smode
SAME| NEXT| SEQuencer| NSEam NSEam = Next Segment Seamless

set_source(*source: TrigSour*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:SOURce
driver.source.bb.arbitrary.trigger.set_source(source = enums.TrigSour.BBSY)


    INTRO_CMD_HELP: Selects the trigger signal source and determines the way␣
↪the triggering is executed. Provided are:

    - Internal triggering by a command (INTernal)

    :param source: INTernal| EXTernal
```

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.trigger.clone()
```

Subgroups

6.18.3.2.6.1 Arm

class ArmCls

Arm 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.bb.arbitrary.trigger.arm.clone()
```

Subgroups

6.18.3.2.6.2 Execute

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:ARM:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:ARM:EXECute
driver.source.bb.arbitrary.trigger.arm.execute.set()
```

Stops (arms) waveform output.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:ARM:EXECute
driver.source.bb.arbitrary.trigger.arm.execute.set_with_opc()
```

Stops (arms) waveform output.

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

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

6.18.3.2.6.3 Delay

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:DELay:UNIT
```

class DelayCls

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

get_unit() → TrigDelUnit

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:DELay:UNIT
value: enums.TrigDelUnit = driver.source.bb.arbitrary.trigger.delay.get_unit()
```

Sets the units the trigger delay is expressed in.

return
trig_del_unit: SAMPle| TIME

set_unit(*trig_del_unit: TrigDelUnit*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:DELay:UNIT
driver.source.bb.arbitrary.trigger.delay.set_unit(trig_del_unit = enums.
↪TrigDelUnit.SAMPle)
```

Sets the units the trigger delay is expressed in.

param trig_del_unit
SAMPle| TIME

6.18.3.2.6.4 Execute

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:EXECute
driver.source.bb.arbitrary.trigger.execute.set()
```

Executes an internal trigger event.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:EXECute
driver.source.bb.arbitrary.trigger.execute.set_with_opc()
```

Executes an internal trigger event.

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

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

6.18.3.2.6.5 External

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:DELay
[SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:INHibit
[SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:RDELay
[SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:TDELay
```

class ExternalCls

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

get_delay() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:DELay
value: float = driver.source.bb.arbitrary.trigger.external.get_delay()
```

Specifies the trigger delay in samples. Maximum trigger delay and trigger inhibit values depend on the installed options. See 'Specifying delay and inhibit values'.

return
delay: float Range: 0 to depends on the sample rate, Unit: sample

get_inhibit() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:INHibit
value: int = driver.source.bb.arbitrary.trigger.external.get_inhibit()
```

Specifies the number of samples, by which a restart is inhibited. Maximum trigger delay and trigger inhibit values depend on the installed options. See 'Specifying delay and inhibit values'.

return
inhibit: integer Range: 0 to 21.47 * (clock frequency) , Unit: samples

get_rdelay() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:RDELay
value: float = driver.source.bb.arbitrary.trigger.external.get_rdelay()
```

Queries the time (in seconds) an external trigger event is delayed for.

return
res_time_delay_sec: float Range: 0 to 688

get_tdelay() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:TDELay
value: float = driver.source.bb.arbitrary.trigger.external.get_tdelay()
```

Specifies the trigger delay for external triggering. The value affects all external trigger signals. Maximum trigger delay and trigger inhibit values depend on the installed options. See 'Specifying delay and inhibit values'.

return
ext_time_delay: float Range: 0 to 2147483647 / (clock frequency) , Unit: s

set_delay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:DELay
driver.source.bb.arbitrary.trigger.external.set_delay(delay = 1.0)
```

Specifies the trigger delay in samples. Maximum trigger delay and trigger inhibit values depend on the installed options. See 'Specifying delay and inhibit values'.

param delay
float Range: 0 to depends on the sample rate, Unit: sample

set_inhibit(*inhibit: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:INHibit
driver.source.bb.arbitrary.trigger.external.set_inhibit(inhibit = 1)
```

Specifies the number of samples, by which a restart is inhibited. Maximum trigger delay and trigger inhibit values depend on the installed options. See 'Specifying delay and inhibit values'.

param inhibit
integer Range: 0 to 21.47 * (clock frequency) , Unit: samples

set_tdelay(*ext_time_delay: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:TDELay
driver.source.bb.arbitrary.trigger.external.set_tdelay(ext_time_delay = 1.0)
```

Specifies the trigger delay for external triggering. The value affects all external trigger signals. Maximum trigger delay and trigger inhibit values depend on the installed options. See 'Specifying delay and inhibit values'.

param ext_time_delay
float Range: 0 to 2147483647 / (clock frequency) , Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.trigger.external.clone()
```

Subgroups

6.18.3.2.6.6 Synchronize

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:SYNChronize:OUTPut
```

class SynchronizeCls

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

get_output() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:SYNChronize:OUTPut
value: bool = driver.source.bb.arbitrary.trigger.external.synchronize.get_
↪output()
```

Enables signal output synchronous to the trigger event.

return
output: 1| ON| 0| OFF

set_output(*output: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:[EXTernal]:SYNChronize:OUTPut
driver.source.bb.arbitrary.trigger.external.synchronize.set_output(output =␣
↪False)
```

Enables signal output synchronous to the trigger event.

param output
1| ON| 0| OFF

6.18.3.2.6.7 Obaseband

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:DELay
[SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:INHibit
[SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:RDELay
```

class ObasebandCls

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

get_delay() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:DELay
value: float = driver.source.bb.arbitrary.trigger.obaseband.get_delay()
```

No command help available

return
delay: No help available

get_inhibit() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:INHibit
value: int = driver.source.bb.arbitrary.trigger.obaseband.get_inhibit()
```

No command help available

return
inhibit: No help available

get_rdelay() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:RDELay
value: float = driver.source.bb.arbitrary.trigger.obaseband.get_rdelay()
```

No command help available

return
res_time_delay_sec: No help available

set_delay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:DELay
driver.source.bb.arbitrary.trigger.obaseband.set_delay(delay = 1.0)
```

No command help available

param delay
No help available

set_inhibit(*inhibit: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OBASeband:INHibit
driver.source.bb.arbitrary.trigger.obaseband.set_inhibit(inhibit = 1)
```

No command help available

param inhibit
No help available

6.18.3.2.6.8 Output<Output>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.arbitrary.trigger.output.repcap_output_get()
driver.source.bb.arbitrary.trigger.output.repcap_output_set(repcap.Output.Nr1)
```

class OutputCls

Output commands group definition. 11 total commands, 7 Subgroups, 0 group commands Repeated Capability: Output, default value after init: Output.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.trigger.output.clone()
```

Subgroups

6.18.3.2.6.9 Delay

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DELay
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut:DELay:FIXed
```

class DelayCls

Delay commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DELay
value: float = driver.source.bb.arbitrary.trigger.output.delay.get(output =␣
↪repcap.Output.Default)
```

Defines the delay between the signal on the marker outputs and the start of the signals.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
delay: integer Range: 0 to depends on other values, Unit: Symbol

get_fixed() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut:DELay:FIXed
value: bool = driver.source.bb.arbitrary.trigger.output.delay.get_fixed()
```

No command help available

return
fixed: No help available

set(*delay: float*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DELay
driver.source.bb.arbitrary.trigger.output.delay.set(delay = 1.0, output =␣
↪repcap.Output.Default)
```

Defines the delay between the signal on the marker outputs and the start of the signals.

param delay
integer Range: 0 to depends on other values, Unit: Symbol

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

set_fixed(*fixed: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut:DELay:FIXed
driver.source.bb.arbitrary.trigger.output.delay.set_fixed(fixed = False)
```

No command help available

param fixed
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.trigger.output.delay.clone()
```

Subgroups

6.18.3.2.6.10 Maximum

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DELay:MAXimum
```

class MaximumCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DELay:MAXimum
value: float = driver.source.bb.arbitrary.trigger.output.delay.maximum.
↪get(output = repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
maximum: No help available

6.18.3.2.6.11 Minimum

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DELay:MINimum
```

class MinimumCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DELay:MINimum
value: float = driver.source.bb.arbitrary.trigger.output.delay.minimum.
↪get(output = repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
minimum: No help available

6.18.3.2.6.12 DinSec

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DINSec
```

class DinSecCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:DINSec
value: float = driver.source.bb.arbitrary.trigger.output.dinSec.get(output =␣
↪repcap.Output.Default)
```

Queries the marker delay in microseconds. You can define a marker delay in samples with [:SOURce<hw>]:BB:ARBitrary:TRIGger:OUTPut<ch>:DELay.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
delay_in_seconds: float Range: 0 to 16777215

6.18.3.2.6.13 Mode

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:MODE
```

class ModeCls

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

get(*output=Output.Default*) → TriggerMarkModeA

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:MODE
value: enums.TriggerMarkModeA = driver.source.bb.arbitrary.trigger.output.mode.
↪get(output = repcap.Output.Default)
```

Defines the signal for the selected marker output. For detailed description of the regular marker modes, refer to 'Marker modes'.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
mode: UNCHanged| RESTart| PULSe| PATTern| RATio UNCHanged A marker signal as defined in the waveform file (tag 'marker mode x') is generated.

set(*mode: TriggerMarkModeA*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:MODE
driver.source.bb.arbitrary.trigger.output.mode.set(mode = enums.
↪TriggerMarkModeA.PATTern, output = repcap.Output.Default)
```

Defines the signal for the selected marker output. For detailed description of the regular marker modes, refer to 'Marker modes'.

param mode
UNCHanged| RESTart| PULSe| PATTern| RATio UNCHanged A marker signal as defined in the waveform file (tag 'marker mode x') is generated.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.2.6.14 OffTime

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:OFFTime
```

class OffTimeCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:OFFTime
value: int = driver.source.bb.arbitrary.trigger.output.offTime.get(output =␣
↪repcap.Output.Default)
```

Sets the number of samples in the ON and OFF periods.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
off_time: integer Range: 1 to 14913079

set(*off_time: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:OFFTime
driver.source.bb.arbitrary.trigger.output.offTime.set(off_time = 1, output =␣
↪repcap.Output.Default)
```

Sets the number of samples in the ON and OFF periods.

param off_time
integer Range: 1 to 14913079

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.2.6.15 Ontime

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:ONTime
```

class OntimeCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:ONTime
value: int = driver.source.bb.arbitrary.trigger.output.ontime.get(output =␣
↪repcap.Output.Default)
```

Sets the number of samples in the ON and OFF periods.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
ontime: No help available

set(*ontime: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:ONTime
driver.source.bb.arbitrary.trigger.output.ontime.set(ontime = 1, output =␣
↪repcap.Output.Default)
```

Sets the number of samples in the ON and OFF periods.

param ontime
integer Range: 1 to 14913079

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.2.6.16 Pattern

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PATTern
```

class PatternCls

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

class PatternStruct

Response structure. Fields:

- Pattern: List[str]: numeric
- Bitcount: int: integer 0 = marker off, 1 = marker on Range: 1 to 64

get(*output=Output.Default*) → PatternStruct

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PATTern
value: PatternStruct = driver.source.bb.arbitrary.trigger.output.pattern.
↪get(output = repcap.Output.Default)
```

Defines the bit pattern used to generate the marker signal.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
structure: for return value, see the help for PatternStruct structure arguments.

set(*pattern: List[str]*, *bitcount: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PATTern
driver.source.bb.arbitrary.trigger.output.pattern.set(pattern = ['rawAbc1',
↪'rawAbc2', 'rawAbc3'], bitcount = 1, output = repcap.Output.Default)
```

Defines the bit pattern used to generate the marker signal.

param pattern
numeric

param bitcount
integer 0 = marker off, 1 = marker on Range: 1 to 64

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.2.6.17 Pulse

class PulseCls

Pulse 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.bb.arbitrary.trigger.output.pulse.clone()
```

Subgroups

6.18.3.2.6.18 Divider

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PULSe:DIVider
```

class DividerCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PULSe:DIVider
value: int = driver.source.bb.arbitrary.trigger.output.pulse.divider.get(output␣
↪= repcap.Output.Default)
```

Sets the divider for the pulsed marker signal.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
divider: integer Range: 2 to 1024

set(*divider: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PULSe:DIVider
driver.source.bb.arbitrary.trigger.output.pulse.divider.set(divider = 1, output␣
↪= repcap.Output.Default)
```

Sets the divider for the pulsed marker signal.

param divider
integer Range: 2 to 1024

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.2.6.19 Frequency

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PULSe:FREQuency
```

class FrequencyCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TRIGger:OUTPut<CH>:PULSe:FREQuency
value: float = driver.source.bb.arbitrary.trigger.output.pulse.frequency.
↪get(output = repcap.Output.Default)
```

Queries the pulse frequency of the pulsed marker signal. The pulse frequency is derived by dividing the symbol rate by the divider.

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
frequency: float

6.18.3.2.7 Tsignal

class TsignalCls

Tsignal commands group definition. 15 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.tsignal.clone()
```

Subgroups

6.18.3.2.7.1 Ciq

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:I
[SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:Q
```

class CiqCls

Ciq commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_icomponent() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:I
value: float = driver.source.bb.arbitrary.tsignal.ciq.get_icomponent()
```

Sets the value for the I and Q component of the test signal

return
ipart: No help available

get_qcomponent() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:Q
value: float = driver.source.bb.arbitrary.tsignal.ciq.get_qcomponent()
```

Sets the value for the I and Q component of the test signal

return
tsig: float Range: -1 to 1, Unit: FS

set_icomponent(*ipart: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:I
driver.source.bb.arbitrary.tsignal.ciq.set_icomponent(ipart = 1.0)
```

Sets the value for the I and Q component of the test signal

param ipart
float Range: -1 to 1, Unit: FS

set_qcomponent(*tsig: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:Q
driver.source.bb.arbitrary.tsignal.ciq.set_qcomponent(tsig = 1.0)
```

Sets the value for the I and Q component of the test signal

param tsig
float Range: -1 to 1, Unit: FS

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.tsignal.ciq.clone()
```

Subgroups

6.18.3.2.7.2 Create

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:CREate
[SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:CREate:NAMed
```

class CreateCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:CREate
driver.source.bb.arbitrary.tsignal.ciq.create.set()
```

Generates a signal and uses it as output straight away.

set_named(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:CREate:NAMed
driver.source.bb.arbitrary.tsignal.ciq.create.set_named(filename = 'abc')
```

Generates a signal and saves it to a waveform file.

param filename
string

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:CIQ:CREate
driver.source.bb.arbitrary.tsignal.ciq.create.set_with_opc()
```

Generates a signal and uses it as output straight away.

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

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

6.18.3.2.7.3 Rectangle

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:AMPLitude
[SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:FREQuency
[SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:OFFSet
[SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:SAMPles
```

class RectangleCls

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

get_amplitude() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:AMPLitude
value: float = driver.source.bb.arbitrary.tsignal.rectangle.get_amplitude()
```

Sets the digital amplitude of the rectangular wave.

return
amplitude: float Range: 0 to 1, Unit: FS

get_frequency() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:FREQuency
value: float = driver.source.bb.arbitrary.tsignal.rectangle.get_frequency()
```

Sets the frequency of the test signal.

return
frequency: float Range: 100 to depends on the installed options, Unit: Hz

get_offset() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:OFFSet
value: float = driver.source.bb.arbitrary.tsignal.rectangle.get_offset()
```

Sets the DC component.

return
offset: float Range: -1 to 1, Unit: FS

get_samples() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:SAMPles
value: int = driver.source.bb.arbitrary.tsignal.rectangle.get_samples()
```

Sets the number of sample values required for the rectangular signal per period.

return
samples: integer Range: 4 to 1000

set_amplitude(*amplitude: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:AMPLitude
driver.source.bb.arbitrary.tsignal.rectangle.set_amplitude(amplitude = 1.0)
```

Sets the digital amplitude of the rectangular wave.

param amplitude
float Range: 0 to 1, Unit: FS

set_frequency(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:FREQuency
driver.source.bb.arbitrary.tsignal.rectangle.set_frequency(frequency = 1.0)
```

Sets the frequency of the test signal.

param frequency
float Range: 100 to depends on the installed options, Unit: Hz

set_offset(*offset: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:OFFSet
driver.source.bb.arbitrary.tsignal.rectangle.set_offset(offset = 1.0)
```

Sets the DC component.

param offset
float Range: -1 to 1, Unit: FS

set_samples(*samples: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:SAMPles
driver.source.bb.arbitrary.tsignal.rectangle.set_samples(samples = 1)
```

Sets the number of sample values required for the rectangular signal per period.

param samples
integer Range: 4 to 1000

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.tsignal.rectangle.clone()
```

Subgroups

6.18.3.2.7.4 Create

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:CREate
[SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:CREate:NAMed
```

class CreateCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:CREate
driver.source.bb.arbitrary.tsignal.rectangle.create.set()
```

Generates a signal and uses it as output straight away.

set_named(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:CREate:NAMed
driver.source.bb.arbitrary.tsignal.rectangle.create.set_named(filename = 'abc')
```

Generates a signal and saves it to a waveform file.

param filename
string

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:RECTangle:CREate
driver.source.bb.arbitrary.tsignal.rectangle.create.set_with_opc()
```

Generates a signal and uses it as output straight away.

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

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

6.18.3.2.7.5 Sine

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:FREQuency
[SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:PHASe
[SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:SAMPles
```

class SineCls

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

get_frequency() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:FREQuency
value: float = driver.source.bb.arbitrary.tsignal.sine.get_frequency()
```

Sets the frequency of the simple sinusoidal test signal.

return
frequency: float Range: 100 to depends on the installed options, Unit: Hz

get_phase() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:PHASe
value: float = driver.source.bb.arbitrary.tsignal.sine.get_phase()
```

Sets the phase offset of the sine wave on the Q channel relative to the sine wave on the I channel.

return
phase: float Range: -180 to 180, Unit: DEG

get_samples() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:SAMPles
value: int = driver.source.bb.arbitrary.tsignal.sine.get_samples()
```

Sets the sample rate for the sine signal in samples per period. The resulting clock rate must not exceed the maximum ARB clock rate (see data sheet) . The maximum value is automatically restricted by reference to the set frequency and has to fulfill the rule Frequency * Samples <= ARB clock rate.

return
samples: integer Range: 4 to 1000

set_frequency(*frequency: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:FREQuency
driver.source.bb.arbitrary.tsignal.sine.set_frequency(frequency = 1.0)
```

Sets the frequency of the simple sinusoidal test signal.

param frequency
float Range: 100 to depends on the installed options, Unit: Hz

set_phase(*phase: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:PHASe
driver.source.bb.arbitrary.tsignal.sine.set_phase(phase = 1.0)
```

Sets the phase offset of the sine wave on the Q channel relative to the sine wave on the I channel.

param phase
float Range: -180 to 180, Unit: DEG

set_samples(*samples: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:SAMPles
driver.source.bb.arbitrary.tsignal.sine.set_samples(samples = 1)
```

Sets the sample rate for the sine signal in samples per period. The resulting clock rate must not exceed the maximum ARB clock rate (see data sheet) . The maximum value is automatically restricted by reference to the set frequency and has to fulfill the rule Frequency * Samples <= ARB clock rate.

param samples
integer Range: 4 to 1000

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.tsignal.sine.clone()
```

Subgroups

6.18.3.2.7.6 Create

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:CREate
[SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:CREate:NAMed
```

class CreateCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:CREate
driver.source.bb.arbitrary.tsignal.sine.create.set()
```

Generates a signal and uses it as output straight away.

set_named(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:CREate:NAMed
driver.source.bb.arbitrary.tsignal.sine.create.set_named(filename = 'abc')
```

Generates a signal and saves it to a waveform file.

param filename
string

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:TSIGnal:SINE:CREate
driver.source.bb.arbitrary.tsignal.sine.create.set_with_opc()
```

Generates a signal and uses it as output straight away.

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

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

6.18.3.2.8 Waveform

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WAVeform:DELete
[SOURce<HW>]:BB:ARBitrary:WAVeform:FREE
[SOURce<HW>]:BB:ARBitrary:WAVeform:POINts
[SOURce<HW>]:BB:ARBitrary:WAVeform:SELect
[SOURce<HW>]:BB:ARBitrary:WAVeform:TAG
```

class WaveformCls

Waveform commands group definition. 9 total commands, 2 Subgroups, 5 group commands

delete(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:DELete
driver.source.bb.arbitrary.waveform.delete(filename = 'abc')
```

Deletes the specified waveform file. If the file is not on the default path, the path must be specified at the same time. The file extension may be omitted. Only files with the file extension *.wv are deleted.

param filename
string

get_free() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:FREE
value: int = driver.source.bb.arbitrary.waveform.get_free()
```

Queries the free disk space on the default path of the instrument's hard disk.

return
free: integer Range: 0 to INT_MAX

get_points() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:POINts
value: int = driver.source.bb.arbitrary.waveform.get_points()
```

Queries the number of samples (the number of I/Q values pairs) in the selected waveform file.

return
points: waveform filename Range: 0 to 1000

get_select() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:SELect
value: str = driver.source.bb.arbitrary.waveform.get_select()
```

Selects an existing waveform file, i.e. file with extension *.wv.

return
filename: string

get_tag() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:TAG
value: str = driver.source.bb.arbitrary.waveform.get_tag()
```

Queries the content of the specified tag of the selected waveform file (see also 'Tags for waveforms, data and control lists') .

return
tag: 'comment'| 'copyright'| 'date'| 'lacpfilter'| 'marker name'| 'poweroffset'| 'samples'

set_select(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:SELect
driver.source.bb.arbitrary.waveform.set_select(filename = 'abc')
```

Selects an existing waveform file, i.e. file with extension *.wv.

param filename
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.waveform.clone()
```

Subgroups

6.18.3.2.8.1 Catalog

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WAVeform:CATalog:LENGth
[SOURce<HW>]:BB:ARBitrary:WAVeform:CATalog
```

class CatalogCls

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

get_length() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:CATalog:LENGth
value: int = driver.source.bb.arbitrary.waveform.catalog.get_length()
```

Reads out the files with extension *.wv in the default directory and returns the number of waveform files in this directory. The default directory is set using command method RsSmcv.MassMemory.currentDirectory.

return
length: integer Number of waveform files in default directory Range: 0 to INT_MAX

get_value() → List[str]

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:CATalog
value: List[str] = driver.source.bb.arbitrary.waveform.catalog.get_value()
```

Reads out the files extension *.wv in the default directory.

return
catalog: string Returns a list of the file names separated by commas

6.18.3.2.8.2 HddStreaming

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WAVeform:HDDStreaming:BLEVel
[SOURce<HW>]:BB:ARBitrary:WAVeform:HDDStreaming:STATe
```

class HddStreamingCls

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

get_blevel() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:HDDStreaming:BLEVel
value: int = driver.source.bb.arbitrary.waveform.hddStreaming.get_blevel()
```

No command help available

return
blevel: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:HDDStreaming:STATe
value: bool = driver.source.bb.arbitrary.waveform.hddStreaming.get_state()
```

By processing large files, enables the streaming of modulation data directly from the hard drive (HDD) .

return
state: 1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WAVeform:HDDStreaming:STATe
driver.source.bb.arbitrary.waveform.hddStreaming.set_state(state = False)
```

By processing large files, enables the streaming of modulation data directly from the hard drive (HDD) .

param state
1| ON| 0| OFF

6.18.3.2.9 Wsegment

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:CLOad
[SOURce<HW>]:BB:ARBitrary:WSEGment:CREate
[SOURce<HW>]:BB:ARBitrary:WSEGment:LMODe
[SOURce<HW>]:BB:ARBitrary:WSEGment:NAME
[SOURce<HW>]:BB:ARBitrary:WSEGment
```

class WsegmentCls

Wsegment commands group definition. 23 total commands, 3 Subgroups, 5 group commands

get_lmode() → ArbLevMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:LMODe
value: enums.ArbLevMode = driver.source.bb.arbitrary.wsegment.get_lmode()
```

Sets how the segments are leveled.

return
level_mode: HIGHest| UNCHanged

get_name() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:NAME
value: str = driver.source.bb.arbitrary.wsegment.get_name()
```

Queries the name of the waveform of the currently output segment of the multi-segment waveform.

return
name: string

get_value() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment
value: int = driver.source.bb.arbitrary.wsegment.get_value()
```

Queries the index of the currently processed segment.

return
wsegment: integer Range: 0 to 1023

set_cload(*filename_input: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CLOad
driver.source.bb.arbitrary.wsegment.set_cload(filename_input = 'abc')
```

Creates a multi-segment waveform using the current entries of the specified configuration file (*.inf_mswv) . *The ARB generator is activated, the new multi-segment waveform (*.wv) is loaded and the first segment is output in accordance to the trigger settings.

param filename_input
string Complete file path, file name of the configuration file and file extension (*.inf_mswv)

set_create(*filename_input: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CREate
driver.source.bb.arbitrary.wsegment.set_create(filename_input = 'abc')
```

Creates a multi-segment waveform (*.wv) *using the current settings of the specified configuration file (*.inf_mswv) .

param filename_input
Complete file path, file name of the configuration file and file extension (*.inf_mswv)

set_lmode(*level_mode: ArbLevMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:LMODe
driver.source.bb.arbitrary.wsegment.set_lmode(level_mode = enums.ArbLevMode.
↪HIGHest)
```

Sets how the segments are leveled.

param level_mode
HIGHest| UNCHanged

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.wsegment.clone()
```

Subgroups

6.18.3.2.9.1 Configure

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CATalog
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:COMMent
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:DELete
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:OFILe
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:SELect
```

class ConfigureCls

Configure commands group definition. 14 total commands, 5 Subgroups, 5 group commands

delete(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:DELete
driver.source.bb.arbitrary.wsegment.configure.delete(filename = 'abc')
```

Deletes the selected configuration file.

param filename
string

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CATalog
value: List[str] = driver.source.bb.arbitrary.wsegment.configure.get_catalog()
```

Queries the available configuration files in the default directory. See also 'File concept'.

return
catalog: string

get_comment() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:COMMent
value: str = driver.source.bb.arbitrary.wsegment.configure.get_comment()
```

Enters a comment for the selected configuration file.

return
comment: string

get_ofile() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:OFILe
value: str = driver.source.bb.arbitrary.wsegment.configure.get_ofile()
```

Defines the file name of the output multi-segment waveform.

return
ofile: string

get_select() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:SELect
value: str = driver.source.bb.arbitrary.wsegment.configure.get_select()
```

Selects a configuration file from the default directory. If a configuration file with the specified name does not yet exist, it is created. The file extension *.inf_mswv may be omitted.

return
filename: string

set_comment(*comment: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:COMMent
driver.source.bb.arbitrary.wsegment.configure.set_comment(comment = 'abc')
```

Enters a comment for the selected configuration file.

param comment
string

set_ofile(*ofile: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:OFILe
driver.source.bb.arbitrary.wsegment.configure.set_ofile(ofile = 'abc')
```

Defines the file name of the output multi-segment waveform.

param ofile
string

set_select(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:SELect
driver.source.bb.arbitrary.wsegment.configure.set_select(filename = 'abc')
```

Selects a configuration file from the default directory. If a configuration file with the specified name does not yet exist, it is created. The file extension *.inf_mswv may be omitted.

param filename
string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.wsegment.configure.clone()
```

Subgroups

6.18.3.2.9.2 Blank

class BlankCls

Blank 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.bb.arbitrary.wsegment.configure.blank.clone()
```

Subgroups

6.18.3.2.9.3 Append

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:BLANk:APPend
```

class AppendCls

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

set(*samp_count: float*, *frequency: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:BLANk:APPend
driver.source.bb.arbitrary.wsegment.configure.blank.append.set(samp_count = 1.0,
↪ frequency = 1.0)
```

Adds a blank segment to the multi-segment file.

param samp_count
float Specifies the number of samples. Range: 512 to 1E7

param frequency
float Determines the clock rate. Range: 400 Hz to depends on the installed options

6.18.3.2.9.4 Clock

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk:MODE
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk
```

class ClockCls

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

get_mode() → ArbWaveSegmClocMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk:MODE
value: enums.ArbWaveSegmClocMode = driver.source.bb.arbitrary.wsegment.
↪configure.clock.get_mode()
```

Selects the clock rate mode for the multi segment waveform. Use the command [:SOURce<hw>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk to define the clock in clock mode user.

return
mode: UNCHanged| HIGHest| USER

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk
value: float = driver.source.bb.arbitrary.wsegment.configure.clock.get_value()
```

Defines the clock rate used for multi-segment waveform output if the clock mode is USER.

return
clock: float

set_mode(*mode: ArbWaveSegmClocMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk:MODE
driver.source.bb.arbitrary.wsegment.configure.clock.set_mode(mode = enums.
↪ArbWaveSegmClocMode.HIGHest)
```

Selects the clock rate mode for the multi segment waveform. Use the command [:SOURce<hw>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk to define the clock in clock mode user.

param mode
UNCHanged| HIGHest| USER

set_value(*clock: float*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:CLOCk
driver.source.bb.arbitrary.wsegment.configure.clock.set_value(clock = 1.0)
```

Defines the clock rate used for multi-segment waveform output if the clock mode is USER.

param clock
float

6.18.3.2.9.5 Level

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:LEVel:[MODE]
```

class LevelCls

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

get_mode() → ArbWaveSegmPowMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:LEVel:[MODE]
value: enums.ArbWaveSegmPowMode = driver.source.bb.arbitrary.wsegment.configure.
↪level.get_mode()
```

Selects the level mode, unchanged or equal RMS, for the multi-segment waveform.

return
mode: UNCHanged| ERMS

set_mode(*mode: ArbWaveSegmPowMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:LEVel:[MODE]
driver.source.bb.arbitrary.wsegment.configure.level.set_mode(mode = enums.
↪ArbWaveSegmPowMode.ERMS)
```

Selects the level mode, unchanged or equal RMS, for the multi-segment waveform.

param mode
UNCHanged| ERMS

6.18.3.2.9.6 Marker

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:ESEGment
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:FSEGment
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:MODE
```

class MarkerCls

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

get_esegment() → ArbWaveSegmRest

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:ESEGment
value: enums.ArbWaveSegmRest = driver.source.bb.arbitrary.wsegment.configure.
↪marker.get_esegment()
```

Enables/disables the generation of an additional marker restart signal at the beginning of the first segment (FSEGment) or at the beginning of each segment (ESEGment) . If additional marker generation is enabled, the existing marker signals in the individual segment waveform files are not considered.

return
mode: OFF| MRK1| MRK2| MRK3| MRK4

get_fsegment() → ArbWaveSegmRest

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:FSEGment
value: enums.ArbWaveSegmRest = driver.source.bb.arbitrary.wsegment.configure.
↪marker.get_fsegment()
```

Enables/disables the generation of an additional marker restart signal at the beginning of the first segment (FSEGment) or at the beginning of each segment (ESEGment) . If additional marker generation is enabled, the existing marker signals in the individual segment waveform files are not considered.

return
mode: OFF| MRK1| MRK2| MRK3| MRK4

get_mode() → ArbWaveSegmMarkMode

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:MODE
value: enums.ArbWaveSegmMarkMode = driver.source.bb.arbitrary.wsegment.
↪configure.marker.get_mode()
```

Defines the way the marker information within the separate segments is processed.

return
mode: IGNore| TAKE

set_esegment(*mode: ArbWaveSegmRest*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:ESEGment
driver.source.bb.arbitrary.wsegment.configure.marker.set_esegment(mode = enums.
↪ArbWaveSegmRest.MRK1)
```

Enables/disables the generation of an additional marker restart signal at the beginning of the first segment (FSEGment) or at the beginning of each segment (ESEGment) . If additional marker generation is enabled, the existing marker signals in the individual segment waveform files are not considered.

param mode
OFF| MRK1| MRK2| MRK3| MRK4

set_fsegment(*mode: ArbWaveSegmRest*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:FSEGment
driver.source.bb.arbitrary.wsegment.configure.marker.set_fsegment(mode = enums.
↪ArbWaveSegmRest.MRK1)
```

Enables/disables the generation of an additional marker restart signal at the beginning of the first segment (FSEGment) or at the beginning of each segment (ESEGment) . If additional marker generation is enabled, the existing marker signals in the individual segment waveform files are not considered.

param mode
OFF| MRK1| MRK2| MRK3| MRK4

set_mode(*mode: ArbWaveSegmMarkMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:MARKer:MODE
driver.source.bb.arbitrary.wsegment.configure.marker.set_mode(mode = enums.
↪ArbWaveSegmMarkMode.IGNore)
```

Defines the way the marker information within the separate segments is processed.

param mode
IGNore| TAKE

6.18.3.2.9.7 Segment

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:SEGMent:APPend
[SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:SEGMent:CATalog
```

class SegmentCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:SEGMent:CATalog
value: List[str] = driver.source.bb.arbitrary.wsegment.configure.segment.get_
↪catalog()
```

Queries the segments of the currently selected configuration file.

return
catalog: string

set_append(*waveform: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:CONFigure:SEGMent:APPend
driver.source.bb.arbitrary.wsegment.configure.segment.set_append(waveform = 'abc
↪')
```

Appends the specified waveform to the configuration file.

param waveform
string

6.18.3.2.9.8 Next

SCPI Commands :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT:SOURce
[SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT
```

class NextCls

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

get_source() → ArbSegmNextSource

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT:SOURce
value: enums.ArbSegmNextSource = driver.source.bb.arbitrary.wsegment.next.get_
↪source()
```

Selects the next segment source.

return
source: INTernal| NSEGM1 | INTernal| NSEGM1

get_value() → int

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT
value: int = driver.source.bb.arbitrary.wsegment.next.get_value()
```

Selects the segment to be output.

return
next_py: integer Range: 0 to 1023

set_source(*source: ArbSegmNextSource*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT:SOURce
driver.source.bb.arbitrary.wsegment.next.set_source(source = enums.
↪ArbSegmNextSource.INTernal)
```

Selects the next segment source.

param source
INTernal| NSEGM1 | INTernal| NSEGM1

set_value(*next_py: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT
driver.source.bb.arbitrary.wsegment.next.set_value(next_py = 1)
```

Selects the segment to be output.

param next_py
integer Range: 0 to 1023

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.arbitrary.wsegment.next.clone()
```

Subgroups

6.18.3.2.9.9 Execute

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT:EXECute
driver.source.bb.arbitrary.wsegment.next.execute.set()
```

Triggers manually switchover to the subsequent segment in the multi-segment file. This command is disabled, if a sequencing play list is enabled.

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

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:NEXT:EXECute
driver.source.bb.arbitrary.wsegment.next.execute.set_with_opc()
```

Triggers manually switchover to the subsequent segment in the multi-segment file. This command is disabled, if a sequencing play list is enabled.

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

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

6.18.3.2.9.10 Sequence

SCPI Command :

```
[SOURce<HW>]:BB:ARBitrary:WSEGment:SEQuence:SELect
```

class SequenceCls

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

get_select() → str

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:SEQuence:SELect
value: str = driver.source.bb.arbitrary.wsegment.sequence.get_select()
```

Selects the sequencing list (files with extension *.wvs)

return
filename: string

set_select(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ARBitrary:WSEGment:SEQuence:SELect
driver.source.bb.arbitrary.wsegment.sequence.set_select(filename = 'abc')
```

Selects the sequencing list (files with extension *.wvs)

param filename
string

6.18.3.3 Atsm

SCPI Commands :

```
[SOURce<HW>]:BB:ATSM:CONStel
[SOURce<HW>]:BB:ATSM:MHEPid
[SOURce<HW>]:BB:ATSM:MHSTate
[SOURce<HW>]:BB:ATSM:PACKetlength
[SOURce<HW>]:BB:ATSM:PAYLoad
[SOURce<HW>]:BB:ATSM:PID
```

(continues on next page)

(continued from previous page)

```
[SOURce<HW>]:BB:ATSM:PIDTestpack
[SOURce<HW>]:BB:ATSM:PRBS
[SOURce<HW>]:BB:ATSM:PRESet
[SOURce<HW>]:BB:ATSM:ROLLoff
[SOURce<HW>]:BB:ATSM:SOURce
[SOURce<HW>]:BB:ATSM:STATe
[SOURce<HW>]:BB:ATSM:STUFfing
[SOURce<HW>]:BB:ATSM:TESTsignal
[SOURce<HW>]:BB:ATSM:TRANsmission
[SOURce<HW>]:BB:ATSM:TSPacket
[SOURce<HW>]:BB:ATSM:WATermark
```

class AtsmCls

Atsm commands group definition. 34 total commands, 9 Subgroups, 17 group commands

get_constel() → AtscmhCodingConstel

```
# SCPI: [SOURce<HW>]:BB:ATSM:CONStel
value: enums.AtscmhCodingConstel = driver.source.bb.atsm.get_constel()
```

Queries the constellation.

return
constellation: VSB8

get_mh_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ATSM:MHSTate
value: bool = driver.source.bb.atsm.get_mh_state()
```

Enables/disableses all ATSC-M/H elements of the .

return
mh_state: 1| ON| 0| OFF ON ATSC-M/H-compliant output signal OFF 8VSB state, output signal complies with the ATSC digital television standard (A/53)

get_mhe_pid() → int

```
# SCPI: [SOURce<HW>]:BB:ATSM:MHEPid
value: int = driver.source.bb.atsm.get_mhe_pid()
```

Sets the PID of MPEG-2 packets that contain ATSC M/H data. The PID is a four-digit value in hexadecimal format.

return
mhe_pid: integer Range: 0 to 8191

get_packet_length() → AtscmhCodingInputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:ATSM:PACKetlength
value: enums.AtscmhCodingInputSignalPacketLength = driver.source.bb.atsm.get_
↪packet_length()
```

Queries the packet length of the external transport stream in bytes.

return
packet_length: P188| P208| INValid P188|P208 188/208 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:ATSM:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.atsm.get_payload()
```

Defines the payload area content of the packet.

return
payload: HFF| H00| PRBS

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:ATSM:PID
value: int = driver.source.bb.atsm.get_pid()
```

Sets the .

return
pid: integer Range: 0 to 8191

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:ATSM:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.atsm.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
pid_test_pack: NULL| VARiable

get_prbs() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:ATSM:PRBS
value: enums.SettingsPrbs = driver.source.bb.atsm.get_prbs()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
prbs: P23_1| P15_1

get_rolloff() → AtscmhCodingRolloff

```
# SCPI: [SOURce<HW>]:BB:ATSM:ROLLoff
value: enums.AtscmhCodingRolloff = driver.source.bb.atsm.get_rolloff()
```

Queries the roll-off factor alpha (alpha) .

return
rolloff: R115

get_source() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:ATSM:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.atsm.get_source()
```

Sets the modulation source for the input signal.

return
atscmh_source: EXTernal| TSPLayer| TESTsignal

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ATSM:STATe
value: bool = driver.source.bb.atsm.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_stuffing() → bool

```
# SCPI: [SOURce<HW>]:BB:ATSM:STUFfing
value: bool = driver.source.bb.atsm.get_stuffing()
```

Activates stuffing.

return
stuffing: 1| ON| 0| OFF

get_test_signal() → AtscmhInputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:ATSM:TESTsignal
value: enums.AtscmhInputSignalTestSignal = driver.source.bb.atsm.get_test_
↪signal()
```

Defines the test signal data.

return
test_signal: TTSP| PBIN| PBET| PBEM TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBIN PRBS before interleaver. Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. PRBS data conforms with specification. PBET PRBS before trellis. Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure and interleaving. Modulation data is directly fed to the trellis encoder. PBEM PRBS before mapper. Pure pseudo-random bit sequence (PRBS) data directly fed to the mapper. Three bits at a time in two's complement are assigned to the stages -7, -5, -3, -1, 1, 3, 5, 7. Subsequent pilot insertion and VSB filtering remain unaffected.

get_transmission() → bool

```
# SCPI: [SOURce<HW>]:BB:ATSM:TRANsmission
value: bool = driver.source.bb.atsm.get_transmission()
```

Enables/disables transmission.

return
transmission: 1| ON| 0| OFF

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:ATSM:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.atsm.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
ts_packet: H184| S187

get_watermark() → bool

```
# SCPI: [SOURce<HW>]:BB:ATSM:WATermark
value: bool = driver.source.bb.atsm.get_watermark()
```

Enables/disables the RF watermark.

return
watermark: 1| ON| 0| OFF

preset() → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:PRESet
driver.source.bb.atsm.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>:BB:ATSM:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:ATSM:PRESet
driver.source.bb.atsm.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>:BB:ATSM:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_mh_state(*mh_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:MHSTate
driver.source.bb.atsm.set_mh_state(mh_state = False)
```

Enables/disableses all ATSC-M/H elements of the .

param mh_state
1| ON| 0| OFF ON ATSC-M/H-compliant output signal OFF 8VSB state, output signal complies with the ATSC digital television standard (A/53)

set_mhe_pid(*mhe_pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:MHEPid
driver.source.bb.atsm.set_mhe_pid(mhe_pid = 1)
```

Sets the PID of MPEG-2 packets that contain ATSC M/H data. The PID is a four-digit value in hexadecimal format.

param mhe_pid
integer Range: 0 to 8191

set_payload(*payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:PAYLoad
driver.source.bb.atsm.set_payload(payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param payload
HFF| H00| PRBS

set_pid(*pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:PID
driver.source.bb.atsm.set_pid(pid = 1)
```

Sets the .

param pid
integer Range: 0 to 8191

set_pid_test_pack(*pid_test_pack: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:PIDTestpack
driver.source.bb.atsm.set_pid_test_pack(pid_test_pack = enums.PidTestPacket.
↪NULL)
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

param pid_test_pack
NULL| VARiable

set_prbs(*prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:PRBS
driver.source.bb.atsm.set_prbs(prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param prbs
P23_1| P15_1

set_source(*atscmh_source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:SOURce
driver.source.bb.atsm.set_source(atscmh_source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param atscmh_source
EXTernal| TSPLayer| TESTsignal

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:STATe
driver.source.bb.atsm.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

set_stuffing(*stuffing: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:STUFfing
driver.source.bb.atsm.set_stuffing(stuffing = False)
```

Activates stuffing.

param stuffing
1| ON| 0| OFF

set_test_signal(*test_signal: AtscmhInputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:TESTsignal
driver.source.bb.atsm.set_test_signal(test_signal = enums.
↪AtscmhInputSignalTestSignal.PBEM)
```

Defines the test signal data.

param test_signal
TTSP| PBIN| PBET| PBEM TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBIN PRBS before interleaver. Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. PRBS data conforms with specification. PBET PRBS before trellis. Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure and interleaving. Modulation data is directly fed to the trellis encoder. PBEM PRBS before mapper. Pure pseudo-random bit sequence (PRBS) data directly fed to the mapper. Three bits at a time in two's complement are assigned to the stages -7, -5, -3, -1, 1, 3, 5, 7. Subsequent pilot insertion and VSB filtering remain unaffected.

set_transmission(*transmission: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:TRANsmission
driver.source.bb.atsm.set_transmission(transmission = False)
```

Enables/disables transmission.

param transmission
1| ON| 0| OFF

set_ts_packet(*ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:TSPacket
driver.source.bb.atsm.set_ts_packet(ts_packet = enums.SettingsTestTsPacket.H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param ts_packet
H184| S187

set_watermark(*watermark: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:WATermark
driver.source.bb.atsm.set_watermark(watermark = False)
```

Enables/disables the RF watermark.

param watermark
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.18.3.3.1 Bury

SCPI Command :

```
[SOURce<HW>]:BB:ATSM:BURY:RATio
```

class BuryCls

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

get_ratio() → AtscmhBuryRatio

```
# SCPI: [SOURce<HW>]:BB:ATSM:BURY:RATio
value: enums.AtscmhBuryRatio = driver.source.bb.atsm.bury.get_ratio()
```

Sets the power with that the watermark is added to the payload signal.

return
market_id: DB21| DB24| DB27| DB30| DB33| DB36| DB39 DBxx Bury ration value ‘xx’ in decibel.

set_ratio(*market_id: AtscmhBuryRatio*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:BURY:RATio
driver.source.bb.atsm.bury.set_ratio(market_id = enums.AtscmhBuryRatio.DB21)
```

Sets the power with that the watermark is added to the payload signal.

param market_id
DB21| DB24| DB27| DB30| DB33| DB36| DB39 DBxx Bury ration value ‘xx’ in decibel.

6.18.3.3.2 Frequency

SCPI Command :

```
[SOURce<HW>]:BB:ATSM:FREQuency:VSBFrequency
```

class FrequencyCls

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

get_vsb_frequency() → AtscmhGeneralVsbFrequency

```
# SCPI: [SOURce<HW>]:BB:ATSM:FREQuency:VSBFrequency
value: enums.AtscmhGeneralVsbFrequency = driver.source.bb.atsm.frequency.get_
↪vsb_frequency()
```

Sets the vestigial sideband (VSB) reference frequency point.

return
vsb_frequency: PILot| CENTer

set_vsb_frequency(*vsb_frequency: AtscmhGeneralVsbFrequency*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:FREQuency:VSBFrequency
driver.source.bb.atsm.frequency.set_vsb_frequency(vsb_frequency = enums.
↪AtscmhGeneralVsbFrequency.CENTer)
```

Sets the vestigial sideband (VSB) reference frequency point.

param vsb_frequency
PILot| CENTer

6.18.3.3.3 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:ATSM:INPut:FORMat
[SOURce<HW>]:BB:ATSM:INPut:TSCHannel
[SOURce<HW>]:BB:ATSM:[INPut]:DATarate
[SOURce<HW>]:BB:ATSM:INPut
```

class InputPyCls

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

get_data_rate() → float

```
# SCPI: [SOURce<HW>]:BB:ATSM:[INPut]:DATarate
value: float = driver.source.bb.atsm.inputPy.get_data_rate()


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
measured_data: float Range: 0 to 999999999

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:ATSM:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.atsm.inputPy.get_format_py()
```

Sets the format of the input signal.

return
input_format: ASI| SMPTE

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:ATSM:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.atsm.inputPy.get_ts_channel()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

return
ts_channel: 1| 2| 3| 4

get_value() → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:ATSM:INPut
value: enums.CodingInputSignalInputA = driver.source.bb.atsm.inputPy.get_value()
```

Sets the external input interface.

return
atscmh_input: TS| IP

set_format_py(*input_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:INPut:FORMat
driver.source.bb.atsm.inputPy.set_format_py(input_format = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param input_format
ASI| SMPTE

set_ts_channel(*ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:INPut:TSCHannel
driver.source.bb.atsm.inputPy.set_ts_channel(ts_channel = enums.NumberA._1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param ts_channel
1| 2| 3| 4

set_value(*atscmh_input: CodingInputSignalInputA*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:INPut
driver.source.bb.atsm.inputPy.set_value(atscmh_input = enums.
↪CodingInputSignalInputA.ASI1)
```

Sets the external input interface.

param atscmh_input
TS| IP

6.18.3.3.4 MtxId

SCPI Commands :

```
[SOURce<HW>]:BB:ATSM:MTXid:MID
[SOURce<HW>]:BB:ATSM:MTXid:TID
```

class MtxIdCls

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

get_mid() → int

```
# SCPI: [SOURce<HW>]:BB:ATSM:MTXid:MID
value: int = driver.source.bb.atsm.mtxId.get_mid()
```

Sets the market ID for the transmission.

return
market_id: integer Range: 0 to 511

get_tid() → int

```
# SCPI: [SOURce<HW>]:BB:ATSM:MTXid:TID
value: int = driver.source.bb.atsm.mtxId.get_tid()
```

Sets the transmitter ID for the MTXID transmission.

return
transmitter_id: integer Range: 0 to 31

set_mid(*market_id: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:MTXid:MID
driver.source.bb.atsm.mtxId.set_mid(market_id = 1)
```

Sets the market ID for the transmission.

param market_id
integer Range: 0 to 511

set_tid(*transmitter_id: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:MTXid:TID
driver.source.bb.atsm.mtxId.set_tid(transmitter_id = 1)
```

Sets the transmitter ID for the MTXID transmission.

param transmitter_id
integer Range: 0 to 31

6.18.3.3.5 Network

SCPI Command :

```
[SOURce<HW>]:BB:ATSM:NETWork:ID
```

class NetworkCls

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

get_id() → int

```
# SCPI: [SOURce<HW>]:BB:ATSM:NETWork:ID
value: int = driver.source.bb.atsm.network.get_id()
```

Sets the network ID for the watermark. The network ID is a three-digit value in hexadecimal format.

return
netw_id: integer Range: 0 to 4095

set_id(*netw_id: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:NETWork:ID
driver.source.bb.atsm.network.set_id(netw_id = 1)
```

Sets the network ID for the watermark. The network ID is a three-digit value in hexadecimal format.

param netw_id
integer Range: 0 to 4095

6.18.3.3.6 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:ATSM:SETTing:CATalog
[SOURce<HW>]:BB:ATSM:SETTing:DELete
[SOURce<HW>]:BB:ATSM:SETTing:LOAD
[SOURce<HW>]:BB:ATSM:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:SETTing:DELete
driver.source.bb.atsm.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.atsm. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:ATSM:SETTing:CATalog
value: List[str] = driver.source.bb.atsm.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.atsm. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
catalog: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:ATSM:SETTing:LOAD
value: str = driver.source.bb.atsm.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.atsm. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: No help available

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:ATSM:SETTing:STORe
value: str = driver.source.bb.atsm.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.atsm) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: No help available

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:SETTing:LOAD
driver.source.bb.atsm.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.atsm. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
'filename' Filename or complete file path; file extension can be omitted.

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:SETTing:STORe
driver.source.bb.atsm.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.atsm) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
'filename' Filename or complete file path

6.18.3.3.7 Symbols

SCPI Command :

```
[SOURce<HW>]:BB:ATSM:SYMBols:[RATE]
```

class SymbolsCls

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

get_rate() → int

```
# SCPI: [SOURce<HW>]:BB:ATSM:SYMBols:[RATE]
value: int = driver.source.bb.atsm.symbols.get_rate()
```

Sets the symbol rate.

return
symbol_rate: integer Range: 10224126 to 11300350

set_rate(*symbol_rate: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:SYMBols:[RATE]
driver.source.bb.atsm.symbols.set_rate(symbol_rate = 1)
```

Sets the symbol rate.

param symbol_rate
integer Range: 10224126 to 11300350

6.18.3.3.8 Tx

SCPI Command :

```
[SOURce<HW>]:BB:ATSM:TX:ADDRess
```

class TxCls

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

get_address() → int

```
# SCPI: [SOURce<HW>]:BB:ATSM:TX:ADDRess
value: int = driver.source.bb.atsm.tx.get_address()
```

Sets the TX address that underlays the RF signal as a watermark.

return
tx_addr: integer Range: 0 to 4095

set_address(*tx_addr: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ATSM:TX:ADDRess
driver.source.bb.atsm.tx.set_address(tx_addr = 1)
```

Sets the TX address that underlays the RF signal as a watermark.

param tx_addr
integer Range: 0 to 4095

6.18.3.3.9 Useful

class UsefulCls

Useful 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.source.bb.atsm.useful.clone()
```

Subgroups

6.18.3.3.9.1 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:ATSM:USEFul:[RATE]:MAX
[SOURce<HW>]:BB:ATSM:USEFul:[RATE]
```

class RateCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:BB:ATSM:USEFul:[RATE]:MAX
value: float = driver.source.bb.atsm.useful.rate.get_max()
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

return
max_usefull: float Range: 0 to 999999999

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:ATSM:USEFul:[RATE]
value: float = driver.source.bb.atsm.useful.rate.get_value()
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

return
usefull_data: float Range: 0 to 999999999

6.18.3.4 Coder

SCPI Command :

```
[SOURce<HW>]:BB:CODer:MODE
```

class CoderCls

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

get_mode() → BbCodMode

```
# SCPI: [SOURce<HW>]:BB:CODer:MODE
value: enums.BbCodMode = driver.source.bb.coder.get_mode()
```

No command help available

return
mode: No help available

set_mode(*mode: BbCodMode*) → None

```
# SCPI: [SOURce<HW>]:BB:CODer:MODE
driver.source.bb.coder.set_mode(mode = enums.BbCodMode.BBIN)
```

No command help available

param mode
No help available

6.18.3.5 Dab

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:EFRames
[SOURce<HW>]:BB:DAB:LDURation
[SOURce<HW>]:BB:DAB:MID
[SOURce<HW>]:BB:DAB:PRESet
[SOURce<HW>]:BB:DAB:SID
[SOURce<HW>]:BB:DAB:STATe
[SOURce<HW>]:BB:DAB:TMODe
```

class DabCls

Dab commands group definition. 58 total commands, 11 Subgroups, 7 group commands

get_eframes() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:EFRames
value: float = driver.source.bb.dab.get_eframes()
```

No command help available

return
eframes: No help available

get_lduration() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:LDURation
value: float = driver.source.bb.dab.get_lduration()
```

No command help available

return
lduration: No help available

get_mid() → int

```
# SCPI: [SOURce<HW>]:BB:DAB:MID
value: int = driver.source.bb.dab.get_mid()
```

No command help available

return
mid: No help available

get_sid() → int

```
# SCPI: [SOURce<HW>]:BB:DAB:SID
value: int = driver.source.bb.dab.get_sid()
```

No command help available

return
sid: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:STATe
value: bool = driver.source.bb.dab.get_state()
```

No command help available

return
state: No help available

get_tmode() → DabTxMode

```
# SCPI: [SOURce<HW>]:BB:DAB:TMODe
value: enums.DabTxMode = driver.source.bb.dab.get_tmode()
```

No command help available

return
tmode: No help available

preset() → None

```
# SCPI: [SOURce<HW>]:BB:DAB:PRESet
driver.source.bb.dab.preset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:DAB:PRESet
driver.source.bb.dab.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_eframes(*eframes: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:EFRames
driver.source.bb.dab.set_eframes(eframes = 1.0)
```

No command help available

param eframes
No help available

set_mid(*mid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:MID
driver.source.bb.dab.set_mid(mid = 1)
```

No command help available

param mid
No help available

set_sid(*sid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:SID
driver.source.bb.dab.set_sid(sid = 1)
```

No command help available

param sid
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:STATe
driver.source.bb.dab.set_state(state = False)
```

No command help available

param state
No help available

set_tmode(*tmode: DabTxMode*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TMODe
driver.source.bb.dab.set_tmode(tmode = enums.DabTxMode.I)
```

No command help available

param tmode
No help available

Cloning the Group

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

Subgroups

6.18.3.5.1 Clock

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:CLOCk:MODE
[SOURce<HW>]:BB:DAB:CLOCk:MULTiplier
[SOURce<HW>]:BB:DAB:CLOCk:SOURce
```

class ClockCls

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

get_mode() → ClockModeUnits

```
# SCPI: [SOURce<HW>]:BB:DAB:CLOCk:MODE
value: enums.ClockModeUnits = driver.source.bb.dab.clock.get_mode()
```

No command help available

return
mode: No help available

get_multiplier() → int

```
# SCPI: [SOURce<HW>]:BB:DAB:CLOCk:MULTiplier
value: int = driver.source.bb.dab.clock.get_multiplier()
```

No command help available

return
multiplier: No help available

get_source() → SourceInt

```
# SCPI: [SOURce<HW>]:BB:DAB:CLOCk:SOURce
value: enums.SourceInt = driver.source.bb.dab.clock.get_source()
```

No command help available

return
source: No help available

set_mode(*mode: ClockModeUnits*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:CLOCk:MODE
driver.source.bb.dab.clock.set_mode(mode = enums.ClockModeUnits.MSAMple)
```

No command help available

param mode
No help available

set_multiplier(*multiplier: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:CLOCk:MULTiplier
driver.source.bb.dab.clock.set_multiplier(multiplier = 1)
```

No command help available

param multiplier
No help available

set_source(*source: SourceInt*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:CLOCk:SOURce
driver.source.bb.dab.clock.set_source(source = enums.SourceInt.EXTernal)
```

No command help available

param source
No help available

6.18.3.5.2 Coder

SCPI Command :

```
[SOURce<HW>]:BB:DAB:CODer:[STATe]
```

class CoderCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:CODer:[STATe]
value: bool = driver.source.bb.dab.coder.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:CODer:[STATe]
driver.source.bb.dab.coder.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.5.3 Data

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:DATA:DSELection
[SOURce<HW>]:BB:DAB:DATA
```

class DataCls

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

get_dselection() → str

```
# SCPI: [SOURce<HW>]:BB:DAB:DATA:DSELection
value: str = driver.source.bb.dab.data.get_dselection()
```

No command help available

return
dselection: No help available

get_value() → DabDataSour

```
# SCPI: [SOURce<HW>]:BB:DAB:DATA
value: enums.DabDataSour = driver.source.bb.dab.data.get_value()
```

No command help available

return
data: No help available

set_dselection(*dselection: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:DATA:DSELection
driver.source.bb.dab.data.set_dselection(dselection = 'abc')
```

No command help available

param dselection
No help available

set_value(*data: DabDataSour*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:DATA
driver.source.bb.dab.data.set_value(data = enums.DabDataSour.ALL0)
```

No command help available

param data
No help available

6.18.3.5.4 Eti

SCPI Command :

```
[SOURce<HW>]:BB:DAB:ETI:CATalog
```

class EtiCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DAB:ETI:CATalog
value: List[str] = driver.source.bb.dab.eti.get_catalog()
```

No command help available

return
catalog: No help available

6.18.3.5.5 FilterPy

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:FILTer:OSAMpling
[SOURce<HW>]:BB:DAB:FILTer:TYPE
```

class FilterPyCls

FilterPy commands group definition. 14 total commands, 3 Subgroups, 2 group commands

get_osampling() → int

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:OSAMpling
value: int = driver.source.bb.dab.filterPy.get_osampling()
```

No command help available

return
osampling: No help available

get_type_py() → DmFilterA

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:TYPE
value: enums.DmFilterA = driver.source.bb.dab.filterPy.get_type_py()
```

No command help available

return
type_py: No help available

set_osampling(*osampling: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:OSAMpling
driver.source.bb.dab.filterPy.set_osampling(osampling = 1)
```

No command help available

param osampling
No help available

set_type_py(*type_py: DmFilterA*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:TYPE
driver.source.bb.dab.filterPy.set_type_py(type_py = enums.DmFilterA.APCO25)
```

No command help available

param type_py
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.filterPy.clone()
```

Subgroups

6.18.3.5.5.1 Ilength

SCPI Command :

```
[SOURce<HW>]:BB:DAB:FILTer:ILENgth
```

class IlengthCls

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

get_value() → int

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:ILENgth
value: int = driver.source.bb.dab.filterPy.ilength.get_value()
```

No command help available

return
ilength: No help available

set_value(*ilength: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:ILENgth
driver.source.bb.dab.filterPy.ilength.set_value(ilength = 1)
```

No command help available

param ilength
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.filterPy.ilength.clone()
```

Subgroups

6.18.3.5.5.2 Auto

SCPI Command :

```
[SOURce<HW>]:BB:DAB:FILTer:ILENgth:AUTO:[STATe]
```

class AutoCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:ILENgth:AUTO:[STATe]
value: bool = driver.source.bb.dab.filterPy.ilength.auto.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:ILENgth:AUTO:[STATe]
driver.source.bb.dab.filterPy.ilength.auto.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.5.5.3 Osamplinng

class OsamplinngCls

Osamplinng 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.bb.dab.filterPy.osamplinng.clone()
```

Subgroups

6.18.3.5.5.4 Auto

SCPI Command :

```
[SOURce<HW>]:BB:DAB:FILTer:OSAMplinng:AUTO:[STATe]
```

class AutoCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:OSAMplinng:AUTO:[STATe]
value: bool = driver.source.bb.dab.filterPy.osamplinng.auto.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:OSAMplinng:AUTO:[STATe]
driver.source.bb.dab.filterPy.osamplinng.auto.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.5.5.5 Parameter

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:FILTer:PARameter:APCO25
[SOURce<HW>]:BB:DAB:FILTer:PARameter:GAUSs
[SOURce<HW>]:BB:DAB:FILTer:PARameter:LPASSEVM
[SOURce<HW>]:BB:DAB:FILTer:PARameter:LPASs
[SOURce<HW>]:BB:DAB:FILTer:PARameter:PGAuss
[SOURce<HW>]:BB:DAB:FILTer:PARameter:RCOSine
[SOURce<HW>]:BB:DAB:FILTer:PARameter:SPHase
```

class ParameterCls

Parameter commands group definition. 9 total commands, 1 Subgroups, 7 group commands

get_apco_25() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:APCO25
value: float = driver.source.bb.dab.filterPy.parameter.get_apco_25()
```

No command help available

return
apco_25: No help available

get_gauss() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:GAUSs
value: float = driver.source.bb.dab.filterPy.parameter.get_gauss()
```

No command help available

return
gauss: No help available

get_lpass() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:LPASs
value: float = driver.source.bb.dab.filterPy.parameter.get_lpass()
```

No command help available

return
lpass: No help available

get_lpass_evm() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:LPASSEVM
value: float = driver.source.bb.dab.filterPy.parameter.get_lpass_evm()
```

No command help available

return
lpass_evm: No help available

get_pgauss() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:PGAuss
value: float = driver.source.bb.dab.filterPy.parameter.get_pgauss()
```

No command help available

return
pgauss: No help available

get_rcosine() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:RCOSine
value: float = driver.source.bb.dab.filterPy.parameter.get_rcosine()
```

No command help available

return
rcosine: No help available

get_sphase() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:SPHase
value: float = driver.source.bb.dab.filterPy.parameter.get_sphase()
```

No command help available

return
sphase: No help available

set_apco_25(*apco_25: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:APCO25
driver.source.bb.dab.filterPy.parameter.set_apco_25(apco_25 = 1.0)
```

No command help available

param apco_25
No help available

set_gauss(*gauss: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:GAUSs
driver.source.bb.dab.filterPy.parameter.set_gauss(gauss = 1.0)
```

No command help available

param gauss
No help available

set_lpass(*lpass: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:LPASs
driver.source.bb.dab.filterPy.parameter.set_lpass(lpass = 1.0)
```

No command help available

param lpass
No help available

set_lpass_evm(*lpass_evm: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:LPASSEVM
driver.source.bb.dab.filterPy.parameter.set_lpass_evm(lpass_evm = 1.0)
```

No command help available

param lpass_evm
No help available

set_pgauss(*pgauss: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:PGAuss
driver.source.bb.dab.filterPy.parameter.set_pgauss(pgauss = 1.0)
```

No command help available

param pgauss
No help available

set_rcosine(*rcosine: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:RCOSine
driver.source.bb.dab.filterPy.parameter.set_rcosine(rcosine = 1.0)
```

No command help available

param rcosine
No help available

set_sphase(*sphase: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:SPHase
driver.source.bb.dab.filterPy.parameter.set_sphase(sphase = 1.0)
```

No command help available

param sphase
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.filterPy.parameter.clone()
```

Subgroups

6.18.3.5.5.6 Cosine

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:FILTer:PARameter:COSine:COFS
[SOURce<HW>]:BB:DAB:FILTer:PARameter:COSine
```

class CosineCls

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

get_cofs() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:COSine:COFS
value: float = driver.source.bb.dab.filterPy.parameter.cosine.get_cofs()
```

No command help available

return
cofs: No help available

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:COSine
value: float = driver.source.bb.dab.filterPy.parameter.cosine.get_value()
```

No command help available

return
cosine: No help available

set_cofs(*cofs: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:COSine:COFS
driver.source.bb.dab.filterPy.parameter.cosine.set_cofs(cofs = 1.0)
```

No command help available

param cofs
No help available

set_value(*cosine: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:FILTer:PARameter:COSine
driver.source.bb.dab.filterPy.parameter.cosine.set_value(cosine = 1.0)
```

No command help available

param cosine
No help available

6.18.3.5.6 Ileaver

SCPI Command :

```
[SOURce<HW>]:BB:DAB:ILEaver:[STATe]
```

class IleaverCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:ILEaver:[STATe]
value: bool = driver.source.bb.dab.ileaver.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:ILEaver:[STATe]
driver.source.bb.dab.ileaver.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.5.7 PnScrambler

SCPI Command :

```
[SOURce<HW>]:BB:DAB:PNSCrambler:[STATe]
```

class PnScramblerCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:PNSCrambler:[STATe]
value: bool = driver.source.bb.dab.pnScrambler.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:PNSCrambler:[STATe]
driver.source.bb.dab.pnScrambler.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.5.8 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:SETTing:CATalog
[SOURce<HW>]:BB:DAB:SETTing:DELete
[SOURce<HW>]:BB:DAB:SETTing:LOAD
```

class SettingCls

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

delete(*file: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:SETTing:DELete
driver.source.bb.dab.setting.delete(file = 'abc')
```

No command help available

param file
No help available

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DAB:SETTing:CATalog
value: List[str] = driver.source.bb.dab.setting.get_catalog()
```

No command help available

return
catalog: No help available

load(*load: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:SETTing:LOAD
driver.source.bb.dab.setting.load(load = 'abc')
```

No command help available

param load
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.setting.clone()
```

Subgroups

6.18.3.5.8.1 Store

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:SETTing:STORe:FAST
[SOURce<HW>]:BB:DAB:SETTing:STORe
```

class StoreCls

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

get_fast() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:SETTing:STORe:FAST
value: bool = driver.source.bb.dab.setting.store.get_fast()
```

No command help available

return
fast: No help available

set_fast(*fast: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:SETTing:STORe:FAST
driver.source.bb.dab.setting.store.set_fast(fast = False)
```

No command help available

param fast
No help available

set_value(*store: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:SETTing:STORe
driver.source.bb.dab.setting.store.set_value(store = 'abc')
```

No command help available

param store
No help available

6.18.3.5.9 SymbolRate

SCPI Command :

```
[SOURce<HW>]:BB:DAB:SRATe:VARiation
```

class SymbolRateCls

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

get_variation() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:SRATe:VARiation
value: float = driver.source.bb.dab.symbolRate.get_variation()
```

No command help available

return
variation: No help available

set_variation(*variation: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:SRATe:VARiation
driver.source.bb.dab.symbolRate.set_variation(variation = 1.0)
```

No command help available

param variation
No help available

6.18.3.5.10 Tii

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TII:[STATe]
```

class TiiCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:TII:[STATe]
value: bool = driver.source.bb.dab.tii.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TII:[STATe]
driver.source.bb.dab.tii.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.5.11 Trigger

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:TRIGger:RMODe
[SOURce<HW>]:BB:DAB:TRIGger:SLENgth
[SOURce<HW>]:BB:DAB:TRIGger:SOURce
[SOURce<HW>]:BB:DAB:[TRIGger]:SEQuence
```

class TriggerCls

Trigger commands group definition. 21 total commands, 5 Subgroups, 4 group commands

get_rmode() → TrigRunMode

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:RMODe
value: enums.TrigRunMode = driver.source.bb.dab.trigger.get_rmode()
```

No command help available

return
rmode: No help available

get_sequence() → DmTrigMode

```
# SCPI: [SOURce<HW>]:BB:DAB:[TRIGger]:SEQuence
value: enums.DmTrigMode = driver.source.bb.dab.trigger.get_sequence()
```

No command help available

return
sequence: No help available

get_slength() → int

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:SLENgth
value: int = driver.source.bb.dab.trigger.get_slength()
```

No command help available

return
slength: No help available

get_source() → TriggerSourceB

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:SOURce
value: enums.TriggerSourceB = driver.source.bb.dab.trigger.get_source()
```

No command help available

return
source: No help available

set_sequence(*sequence: DmTrigMode*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:[TRIGger]:SEQuence
driver.source.bb.dab.trigger.set_sequence(sequence = enums.DmTrigMode.AAUTo)
```

No command help available

param sequence
No help available

set_slength(*slength: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:SLENgth
driver.source.bb.dab.trigger.set_slength(slength = 1)
```

No command help available

param slength
No help available

set_source(*source: TriggerSourceB*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:SOURce
driver.source.bb.dab.trigger.set_source(source = enums.TriggerSourceB.BEXTernal)
```

No command help available

param source
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.trigger.clone()
```

Subgroups

6.18.3.5.11.1 Arm

class ArmCls

Arm 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.bb.dab.trigger.arm.clone()
```

Subgroups

6.18.3.5.11.2 Execute

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:ARM:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:ARM:EXECute
driver.source.bb.dab.trigger.arm.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:ARM:EXECute
driver.source.bb.dab.trigger.arm.execute.set_with_opc()
```

No command help available

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

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

6.18.3.5.11.3 Execute

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:EXECute
driver.source.bb.dab.trigger.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:EXECute
driver.source.bb.dab.trigger.execute.set_with_opc()
```

No command help available

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

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

6.18.3.5.11.4 External<External>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.source.bb.dab.trigger.external.repcap_external_get()
driver.source.bb.dab.trigger.external.repcap_external_set(repcap.External.Nr1)
```

class ExternalCls

External commands group definition. 3 total commands, 3 Subgroups, 0 group commands Repeated Capability: External, default value after init: External.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.trigger.external.clone()
```

Subgroups

6.18.3.5.11.5 Delay

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:[EXTernal<CH>]:DELay
```

class DelayCls

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

get(*external=External.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:[EXTernal<CH>]:DELay
value: float = driver.source.bb.dab.trigger.external.delay.get(external =␣
↪repcap.External.Default)
```

No command help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

return
delay: No help available

set(*delay: float*, *external=External.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:[EXTernal<CH>]:DELay
driver.source.bb.dab.trigger.external.delay.set(delay = 1.0, external = repcap.
↪External.Default)
```

No command help available

param delay
No help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

6.18.3.5.11.6 Inhibit

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:[EXTernal<CH>]:INHibit
```

class InhibitCls

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

get(*external=External.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:[EXTernal<CH>]:INHibit
value: int = driver.source.bb.dab.trigger.external.inhibit.get(external =␣
↪repcap.External.Default)
```

No command help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

return
inhibit: No help available

set(*inhibit: int*, *external=External.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:[EXTernal<CH>]:INHibit
driver.source.bb.dab.trigger.external.inhibit.set(inhibit = 1, external =␣
↪repcap.External.Default)
```

No command help available

param inhibit
No help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

6.18.3.5.11.7 Synchronize

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:EXTernal:SYNChronize:OUTPut
```

class SynchronizeCls

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

get_output() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:EXTernal:SYNChronize:OUTPut
value: bool = driver.source.bb.dab.trigger.external.synchronize.get_output()
```

No command help available

return
output: No help available

set_output(*output: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:EXTernal:SYNChronize:OUTPut
driver.source.bb.dab.trigger.external.synchronize.set_output(output = False)
```

No command help available

param output
No help available

6.18.3.5.11.8 Obaseband

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:TRIGger:OBASeband:DELay
[SOURce<HW>]:BB:DAB:TRIGger:OBASeband:INHibit
```

class ObasebandCls

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

get_delay() → float

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OBASeband:DELay
value: float = driver.source.bb.dab.trigger.obaseband.get_delay()
```

No command help available

return
delay: No help available

get_inhibit() → int

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OBASeband:INHibit
value: int = driver.source.bb.dab.trigger.obaseband.get_inhibit()
```

No command help available

return
inhibit: No help available

set_delay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OBASeband:DELay
driver.source.bb.dab.trigger.obaseband.set_delay(delay = 1.0)
```

No command help available

param delay
No help available

set_inhibit(*inhibit: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OBASeband:INHibit
driver.source.bb.dab.trigger.obaseband.set_inhibit(inhibit = 1)
```

No command help available

param inhibit
No help available

6.18.3.5.11.9 Output<Output>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.dab.trigger.output.repcap_output_get()
driver.source.bb.dab.trigger.output.repcap_output_set(repcap.Output.Nr1)
```

class OutputCls

Output commands group definition. 10 total commands, 6 Subgroups, 0 group commands Repeated Capability: Output, default value after init: Output.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.trigger.output.clone()
```

Subgroups

6.18.3.5.11.10 Delay

SCPI Commands :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:DELay
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut:DELay:FIXed
```

class DelayCls

Delay commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:DELay
value: float = driver.source.bb.dab.trigger.output.delay.get(output = repcap.
↪Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
delay: No help available

get_fixed() → bool

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut:DELay:FIXed
value: bool = driver.source.bb.dab.trigger.output.delay.get_fixed()
```

No command help available

return
fixed: No help available

set(*delay: float*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:DELay
driver.source.bb.dab.trigger.output.delay.set(delay = 1.0, output = repcap.
↪Output.Default)
```

No command help available

param delay
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

set_fixed(*fixed: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut:DELay:FIXed
driver.source.bb.dab.trigger.output.delay.set_fixed(fixed = False)
```

No command help available

param fixed
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dab.trigger.output.delay.clone()
```

Subgroups

6.18.3.5.11.11 Maximum

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:DELay:MAXimum
```

class MaximumCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:DELay:MAXimum
value: float = driver.source.bb.dab.trigger.output.delay.maximum.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
maximum: No help available

6.18.3.5.11.12 Minimum

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:DELay:MINimum
```

class MinimumCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:DELay:MINimum
value: float = driver.source.bb.dab.trigger.output.delay.minimum.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
minimum: No help available

6.18.3.5.11.13 Mode

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:MODE
```

class ModeCls

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

get(*output=Output.Default*) → MarkModeA

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:MODE
value: enums.MarkModeA = driver.source.bb.dab.trigger.output.mode.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
mode: No help available

set(*mode: MarkModeA*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:MODE
driver.source.bb.dab.trigger.output.mode.set(mode = enums.MarkModeA.FRAMe,␣
↪output = repcap.Output.Default)
```

No command help available

param mode
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.5.11.14 OffTime

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:OFFTime
```

class OffTimeCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:OFFTime
value: int = driver.source.bb.dab.trigger.output.offTime.get(output = repcap.
↪Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
off_time: No help available

set(*off_time: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:OFFTime
driver.source.bb.dab.trigger.output.offTime.set(off_time = 1, output = repcap.
↪Output.Default)
```

No command help available

param off_time
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.5.11.15 Ontime

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:ONTime
```

class OntimeCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:ONTime
value: int = driver.source.bb.dab.trigger.output.ontime.get(output = repcap.
↪Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
ontime: No help available

set(*ontime: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:ONTime
driver.source.bb.dab.trigger.output.ontime.set(ontime = 1, output = repcap.
↪Output.Default)
```

No command help available

param ontime
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.5.11.16 Pattern

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PATTern
```

class PatternCls

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

class PatternStruct

Response structure. Fields:

- Pattern: List[str]: No parameter help available
- Bitcount: int: No parameter help available

get(*output=Output.Default*) → PatternStruct

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PATTern
value: PatternStruct = driver.source.bb.dab.trigger.output.pattern.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
structure: for return value, see the help for PatternStruct structure arguments.

set(*pattern: List[str]*, *bitcount: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PATTern
driver.source.bb.dab.trigger.output.pattern.set(pattern = ['rawAbc1', 'rawAbc2',
↪ 'rawAbc3'], bitcount = 1, output = repcap.Output.Default)
```

No command help available

param pattern
No help available

param bitcount
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.5.11.17 Pulse

class PulseCls

Pulse 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.bb.dab.trigger.output.pulse.clone()
```

Subgroups

6.18.3.5.11.18 Divider

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PULSe:DIVider
```

class DividerCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PULSe:DIVider
value: int = driver.source.bb.dab.trigger.output.pulse.divider.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
divider: No help available

set(*divider: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PULSe:DIVider
driver.source.bb.dab.trigger.output.pulse.divider.set(divider = 1, output =␣
↪repcap.Output.Default)
```

No command help available

param divider
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.5.11.19 Frequency

SCPI Command :

```
[SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PULSe:FREQuency
```

class FrequencyCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DAB:TRIGger:OUTPut<CH>:PULSe:FREQuency
value: float = driver.source.bb.dab.trigger.output.pulse.frequency.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
frequency: No help available

6.18.3.6 Drm

SCPI Commands :

```
[SOURce<HW>]:BB:DRM:BANDwidth
[SOURce<HW>]:BB:DRM:FILename
[SOURce<HW>]:BB:DRM:INTerleaver
[SOURce<HW>]:BB:DRM:LABel
[SOURce<HW>]:BB:DRM:MODE
[SOURce<HW>]:BB:DRM:NUMData
[SOURce<HW>]:BB:DRM:NUMaudio
[SOURce<HW>]:BB:DRM:PORT
[SOURce<HW>]:BB:DRM:PRESet
[SOURce<HW>]:BB:DRM:SOURce
[SOURce<HW>]:BB:DRM:STATe
[SOURce<HW>]:BB:DRM:TYPE
```

class DrmCls

Drm commands group definition. 24 total commands, 3 Subgroups, 12 group commands

get_bandwidth() → DrmCodingChannelBw

```
# SCPI: [SOURce<HW>]:BB:DRM:BANDwidth
value: enums.DrmCodingChannelBw = driver.source.bb.drm.get_bandwidth()
```

Queries the channel bandwidth.

return
drm_bandwidth: K045| K05| K09| K10| K18| K20| K100| INV K045 4.5 kHz K05 5 kHz K09 9 kHz K10 10 kHz K18 18 kHz K20 20 kHz K100 100 kHz INV Invalid channel bandwidth

get_filename() → str

```
# SCPI: [SOURce<HW>]:BB:DRM:FILename
value: str = driver.source.bb.drm.get_filename()
```

Loads the specified file. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
drm_dcp_file: string Filename or complete file path; file extension (*.dcp) can be omitted.

get_interleaver() → DrmCodingInterleaver

```
# SCPI: [SOURce<HW>]:BB:DRM:INTerleaver
value: enums.DrmCodingInterleaver = driver.source.bb.drm.get_interleaver()
```

Queries the interleaver depth.

return
drm_interleaver: MS4| MS6| S2| INV MS4 400 ms MS6 600 ms S2 2 s INV Invalid interleaver depth

get_label() → str

```
# SCPI: [SOURce<HW>]:BB:DRM:LABel
value: str = driver.source.bb.drm.get_label()
```

Queries the label of the transmitted service.

return
drm_label: string Each service has a maximum length of 16 characters separated by br.

get_mode() → DrmCodingRobustness

```
# SCPI: [SOURce<HW>]:BB:DRM:MODE
value: enums.DrmCodingRobustness = driver.source.bb.drm.get_mode()
```

Queries the robustness mode of the signal.

return
drm_mode: A| B| C| D| E| INV A|B|C|D|E Available robustness modes. INV Invalid mode.

get_num_audio() → DrmInputSignalServices

```
# SCPI: [SOURce<HW>]:BB:DRM:NUMaudio
value: enums.DrmInputSignalServices = driver.source.bb.drm.get_num_audio()
```

Queries the number of audio services contained in the input stream.

return
drm_num_audio: 0| 1| 2| 3| 4| INV 0|1|2|3|4 Available number of audio services INV Invalid number of audio services

get_num_data() → DrmInputSignalServices

```
# SCPI: [SOURce<HW>]:BB:DRM:NUMData
value: enums.DrmInputSignalServices = driver.source.bb.drm.get_num_data()
```

Queries the number of data services contained in the input stream.

return
drm_num_data: 0| 1| 2| 3| 4| INV 0|1|2|3|4 Available number of data services INV Invalid number of data services

get_port() → int

```
# SCPI: [SOURce<HW>]:BB:DRM:PORT
value: int = driver.source.bb.drm.get_port()
```

Sets the port. Enter the port number on that the UDP/ receiver listens for UDP datagrams.

return
drm_port: integer Range: 0 to 65535

get_source() → DrmInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DRM:SOURce
value: enums.DrmInputSignalSource = driver.source.bb.drm.get_source()
```

Sets the modulation source for the input signal.

return
drm_source: EXTernal| FILE EXTernal Uses a / stream input at the 'LAN' connector of the host PC of the R&S SMCV100B. FILE Reads the input stream from a *.dcp file. The binary file contains the MDI data encapsulated in packets.

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DRM:STATe
value: bool = driver.source.bb.drm.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_type_py() → DrmInputSignalLayerType

```
# SCPI: [SOURce<HW>]:BB:DRM:TYPE
value: enums.DrmInputSignalLayerType = driver.source.bb.drm.get_type_py()
```

Queries the type of audio in the transmission.

return
drm_layer_type: BASE| ENHancement| INV BASE Decodable by all DRM receivers. ENHancement Only decodable by receivers with appropriate capabilities. INV Invalid type

preset() → None

```
# SCPI: [SOURce<HW>]:BB:DRM:PRESet
driver.source.bb.drm.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>:BB:DRM:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:DRM:PRESet
driver.source.bb.drm.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>:BB:DRM:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_filename(*drm_dcp_file: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DRM:FILename
driver.source.bb.drm.set_filename(drm_dcp_file = 'abc')
```

Loads the specified file. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param drm_dcp_file
string Filename or complete file path; file extension (*.dcp) can be omitted.

set_port(*drm_port: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DRM:PORT
driver.source.bb.drm.set_port(drm_port = 1)
```

Sets the port. Enter the port number on that the UDP/ receiver listens for UDP datagrams.

param drm_port
integer Range: 0 to 65535

set_source(*drm_source: DrmInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:DRM:SOURce
driver.source.bb.drm.set_source(drm_source = enums.DrmInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param drm_source
EXTernal| FILE EXTernal Uses a / stream input at the 'LAN' connector of the host PC of the R&S SMCV100B. FILE Reads the input stream from a *.dcp file. The binary file contains the MDI data encapsulated in packets.

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DRM:STATe
driver.source.bb.drm.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.18.3.6.1 Msc

SCPI Command :

```
[SOURce<HW>]:BB:DRM:MSC:CONStel
```

class MscCls

Msc commands group definition. 4 total commands, 3 Subgroups, 1 group commands

get_constel() → DrmCodingConstelMsc

```
# SCPI: [SOURce<HW>]:BB:DRM:MSC:CONStel
value: enums.DrmCodingConstelMsc = driver.source.bb.drm.msc.get_constel()
```

Queries the constellation of the .

return
drm_const_msc: Q64N| Q64I| Q64Q| Q16| Q4| INV Q64N 64 non-hierarchical Q64I 64QAM hierarchical on I Q64Q 64QAM hierarchical on I and Q Q16 16QAM non-hierarchical Q4 4QAM non-hierarchical INV Invalid constellation

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.drm.msc.clone()
```

Subgroups

6.18.3.6.1.1 Level<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.drm.msc.level.repcap_index_get()
driver.source.bb.drm.msc.level.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:DRM:MSC:LEVel<CH>
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → DrmCodingProtectionLevelMsc

```
# SCPI: [SOURce<HW>]:BB:DRM:MSC:LEVel<CH>
value: enums.DrmCodingProtectionLevelMsc = driver.source.bb.drm.msc.level.
↪get(index = repcap.Index.Default)
```

Queries the protection level used in each of the protection profiles.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Level')

return
drm_msc_lev: 0| 1| 2| 3| INV 0|1|2|3 Available protection levels INV Invalid protection level

Cloning the Group

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

6.18.3.6.1.2 Profile<Profile>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.bb.drm.msc.profile.repcap_profile_get()
driver.source.bb.drm.msc.profile.repcap_profile_set(repcap.Profile.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:DRM:MSC:PROFile<CH>
```

class ProfileCls

Profile commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Profile, default value after init: Profile.Nr1

get(*profile=Profile.Default*) → DrmCodingProtectionProfileMsc

```
# SCPI: [SOURce<HW>]:BB:DRM:MSC:PROFile<CH>
value: enums.DrmCodingProtectionProfileMsc = driver.source.bb.drm.msc.profile.
↪get(profile = repcap.Profile.Default)
```

Queries the protection profile used in the transmission.

param profile
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Profile')

return
drm_msc_prof: HPP| LPP| VSPP| INV HPP Higher protected part LPP Lower protected part VSPP Very strongly protected part INV Invalid protection profile

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.drm.msc.profile.clone()
```

6.18.3.6.1.3 Rate<Profile>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.bb.drm.msc.rate.repcap_profile_get()
driver.source.bb.drm.msc.rate.repcap_profile_set(repcap.Profile.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:DRM:MSC:RATE<CH>
```

class RateCls

Rate commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Profile, default value after init: Profile.Nr1

get(*profile=Profile.Default*) → DrmCodingCoderate

```
# SCPI: [SOURce<HW>]:BB:DRM:MSC:RATE<CH>
value: enums.DrmCodingCoderate = driver.source.bb.drm.msc.rate.get(profile =␣
↪repcap.Profile.Default)
```

Queries the overall code rate used in each of the protection profiles.

param profile
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rate')

return
drm_msc_rate: R025| R033| R040| R041| R045| R048| R050| R055| R057| R058| R060| R062| R066| R071| R072| R078| INV R0xy 0xy constitutes a code rate of 0.xy INV Invalid code rate

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.drm.msc.rate.clone()
```

6.18.3.6.2 Sdc

SCPI Command :

```
[SOURce<HW>]:BB:DRM:SDC:CONStel
```

class SdcCls

Sdc commands group definition. 4 total commands, 3 Subgroups, 1 group commands

get_constel() → DrmCodingConstelSdc

```
# SCPI: [SOURce<HW>]:BB:DRM:SDC:CONStel
value: enums.DrmCodingConstelSdc = driver.source.bb.drm.sdc.get_constel()
```

Queries the constellation of the .

return
drm_const_sdc: Q16| Q4| INV Q16 16 Q4 4 INV Invalid constellation

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.drm.sdc.clone()
```

Subgroups

6.18.3.6.2.1 Level<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.drm.sdc.level.repcap_index_get()
driver.source.bb.drm.sdc.level.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:DRM:SDC:LEVel<CH>
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → DrmCodingProtectionLevelSdc

```
# SCPI: [SOURce<HW>]:BB:DRM:SDC:LEVel<CH>
value: enums.DrmCodingProtectionLevelSdc = driver.source.bb.drm.sdc.level.
↪get(index = repcap.Index.Default)
```

Queries the protection level of the .

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Level')

return
drm_sdc_lev: 0| 1| INV 0|1 Available protection levels INV Invalid protection level

Cloning the Group

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

6.18.3.6.2.2 Profile<Profile>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.bb.drm.sdc.profile.repcap_profile_get()
driver.source.bb.drm.sdc.profile.repcap_profile_set(repcap.Profile.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:DRM:SDC:PROFile<CH>
```

class ProfileCls

Profile commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Profile, default value after init: Profile.Nr1

get(*profile=Profile.Default*) → DrmCodingProtectionProfileSdc

```
# SCPI: [SOURce<HW>]:BB:DRM:SDC:PROFile<CH>
value: enums.DrmCodingProtectionProfileSdc = driver.source.bb.drm.sdc.profile.
↪get(profile = repcap.Profile.Default)
```

Queries the protection profile of the .

param profile
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Profile')

return
drm_sdc_prof: EEP| INV EEP Equal error protection INV Invalid protection profile

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.drm.sdc.profile.clone()
```

6.18.3.6.2.3 Rate<Profile>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.bb.drm.sdc.rate.repcap_profile_get()
driver.source.bb.drm.sdc.rate.repcap_profile_set(repcap.Profile.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:DRM:SDC:RATE<CH>
```

class RateCls

Rate commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Profile, default value after init: Profile.Nr1

get(*profile=Profile.Default*) → DrmCodingCoderate

```
# SCPI: [SOURce<HW>]:BB:DRM:SDC:RATE<CH>
value: enums.DrmCodingCoderate = driver.source.bb.drm.sdc.rate.get(profile =␣
↪repcap.Profile.Default)
```

Queries the overall code rate of the .

param profile
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rate')

return
drm_sdc_rate: R025| R033| R040| R041| R045| R048| R050| R055| R057| R058| R060| R062| R066| R071| R072| R078| INV R0xy 0xy constitutes a code rate of 0.xy INV Invalid code rate

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.drm.sdc.rate.clone()
```

6.18.3.6.3 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:DRM:SETTing:CATalog
[SOURce<HW>]:BB:DRM:SETTing:DELete
[SOURce<HW>]:BB:DRM:SETTing:LOAD
[SOURce<HW>]:BB:DRM:SETTing:STORe
```

class SettingCls

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

delete(*drm_delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DRM:SETTing:DELete
driver.source.bb.drm.setting.delete(drm_delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.drm. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param drm_delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DRM:SETTing:CATalog
value: List[str] = driver.source.bb.drm.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.drm. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
drm_cat: filename1,filename2,... Returns a string of filenames separated by commas.

load(*drm_recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DRM:SETTing:LOAD
driver.source.bb.drm.setting.load(drm_recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.drm. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param drm_recall
'DrmRecall' Filename or complete file path; file extension can be omitted

set_store(*drm_save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DRM:SETTing:STORe
driver.source.bb.drm.setting.set_store(drm_save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.drm) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param drm_save
'filename' Filename or complete file path

6.18.3.7 Dtmb

SCPI Commands :

```
[SOURce<HW>]:BB:DTMB:CONStel
[SOURce<HW>]:BB:DTMB:FRAMes
[SOURce<HW>]:BB:DTMB:GIC
[SOURce<HW>]:BB:DTMB:GUARd
[SOURce<HW>]:BB:DTMB:PACKetlength
[SOURce<HW>]:BB:DTMB:PAYLoad
[SOURce<HW>]:BB:DTMB:PID
[SOURce<HW>]:BB:DTMB:PIDTestpack
[SOURce<HW>]:BB:DTMB:PRESet
[SOURce<HW>]:BB:DTMB:RATE
[SOURce<HW>]:BB:DTMB:SINGle
[SOURce<HW>]:BB:DTMB:SOURce
[SOURce<HW>]:BB:DTMB:STATe
[SOURce<HW>]:BB:DTMB:STUFfing
[SOURce<HW>]:BB:DTMB:TESTsignal
[SOURce<HW>]:BB:DTMB:TSPacket
```

class DtmbCls

Dtmb commands group definition. 32 total commands, 8 Subgroups, 16 group commands

get_constel() → DtmbCodingConstel

```
# SCPI: [SOURce<HW>]:BB:DTMB:CONStel
value: enums.DtmbCodingConstel = driver.source.bb.dtmb.get_constel()
```

Defines the constellation.

return
dtmb_constel: D4| D4NR| D16| D32| D64 D4 4QAM D4NR 4QAM-NR D16 16QAM D32 32QAM D64 64QAM

get_frames() → bool

```
# SCPI: [SOURce<HW>]:BB:DTMB:FRAMes
value: bool = driver.source.bb.dtmb.get_frames()
```

Defines whether a control frame is added to each signal frame group or not.

return
dtmb_frames: 1| ON| 0| OFF

get_gic() → DtmbCodingGipN

```
# SCPI: [SOURce<HW>]:BB:DTMB:GIC
value: enums.DtmbCodingGipN = driver.source.bb.dtmb.get_gic()
```

Defines the initial condition of the PN sequences in the frame headers.

return
dtmb_guard_pn: VAR| CONSt VAR Uses the definition of the table in the standard. CONSt Uses the initial condition of index 0 for all signal frames.

get_guard() → DtmbCodingGuardInterval

```
# SCPI: [SOURce<HW>]:BB:DTMB:GUARd
value: enums.DtmbCodingGuardInterval = driver.source.bb.dtmb.get_guard()
```

Sets the guard interval length.

return
dtmb_guard: G420| G595| G945

get_packet_length() → DtmbCodingInputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:DTMB:PACKetlength
value: enums.DtmbCodingInputSignalPacketLength = driver.source.bb.dtmb.get_
↪packet_length()
```

Queries the packet length of the external transport stream in bytes.

return
dtmb_plength: P188| INValid P188 188 byte packets specified for serial input (‘Input TS IN’) and parallel input (‘Input IP’) . INValid Packet length does not match the specified length.

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:DTMB:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.dtmb.get_payload()
```

Defines the payload area content of the packet.

return
dtmb_payload: H00| HFF| PRBS

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:DTMB:PID
value: int = driver.source.bb.dtmb.get_pid()
```

Sets the .

return
dtmb_pid: integer Range: #H0000 to #H1FFF

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:DTMB:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.dtmb.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
dtmb_source: NULL| VARiable

get_rate() → DtmbCodingCoderate

```
# SCPI: [SOURce<HW>]:BB:DTMB:RATE
value: enums.DtmbCodingCoderate = driver.source.bb.dtmb.get_rate()
```

Sets the code rate.

return
dtmb_code_rate: R04| R06| R08

get_single() → bool

```
# SCPI: [SOURce<HW>]:BB:DTMB:SINGle
value: bool = driver.source.bb.dtmb.get_single()
```

Enables/disables single carrier mode.

return
dtmb_single: 1| ON| 0| OFF

get_source() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DTMB:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.dtmb.get_source()
```

Sets the modulation source for the input signal.

return
dtmb_source: EXTernal| TSPLayer| TESTsignal

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DTMB:STATe
value: bool = driver.source.bb.dtmb.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_stuffing() → bool

```
# SCPI: [SOURce<HW>]:BB:DTMB:STUFfing
value: bool = driver.source.bb.dtmb.get_stuffing()
```

Activates stuffing.

return
dtmb_stuffing: 1| ON| 0| OFF

get_test_signal() → CodingInputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:DTMB:TESTsignal
value: enums.CodingInputSignalTestSignal = driver.source.bb.dtmb.get_test_
↪signal()
```

Queries the test signal, that consists of test packets.

return
dtmb_test_sig: TTSP Test TS packet with standardized packet data used as modulation data in the transport stream.

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:DTMB:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.dtmb.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
dtmb_test_ts_pack: H184| S187

preset() → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:PRESet
driver.source.bb.dtmb.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>:BB:DTMB:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:DTMB:PRESet
driver.source.bb.dtmb.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>:BB:DTMB:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_constel(*dtmb_constel: DtmbCodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:CONStel
driver.source.bb.dtmb.set_constel(dtmb_constel = enums.DtmbCodingConstel.D16)
```

Defines the constellation.

param dtmb_constel
D4| D4NR| D16| D32| D64 D4 4QAM D4NR 4QAM-NR D16 16QAM D32 32QAM D64 64QAM

set_frames(*dtmb_frames: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:FRAMes
driver.source.bb.dtmb.set_frames(dtmb_frames = False)
```

Defines whether a control frame is added to each signal frame group or not.

param dtmb_frames
1| ON| 0| OFF

set_gic(*dtmb_guard_pn: DtmbCodingGipN*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:GIC
driver.source.bb.dtmb.set_gic(dtmb_guard_pn = enums.DtmbCodingGipN.CONSt)
```

Defines the initial condition of the PN sequences in the frame headers.

param dtmb_guard_pn
VAR| CONSt VAR Uses the definition of the table in the standard. CONSt Uses the initial condition of index 0 for all signal frames.

set_guard(*dtmb_guard: DtmbCodingGuardInterval*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:GUARd
driver.source.bb.dtmb.set_guard(dtmb_guard = enums.DtmbCodingGuardInterval.G420)
```

Sets the guard interval length.

param dtmb_guard
G420| G595| G945

set_payload(*dtmb_payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:PAYLoad
driver.source.bb.dtmb.set_payload(dtmb_payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param dtmb_payload
H00| HFF| PRBS

set_pid(*dtmb_pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:PID
driver.source.bb.dtmb.set_pid(dtmb_pid = 1)
```

Sets the .

param dtmb_pid
integer Range: #H0000 to #H1FFF

set_pid_test_pack(*dtmb_source: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:PIDTestpack
driver.source.bb.dtmb.set_pid_test_pack(dtmb_source = enums.PidTestPacket.NULL)
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

param dtmb_source
NULL| VARiable

set_rate(*dtmb_code_rate: DtmbCodingCoderate*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:RATE
driver.source.bb.dtmb.set_rate(dtmb_code_rate = enums.DtmbCodingCoderate.R04)
```

Sets the code rate.

param dtmb_code_rate
R04| R06| R08

set_single(*dtmb_single: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:SINGle
driver.source.bb.dtmb.set_single(dtmb_single = False)
```

Enables/disables single carrier mode.

param dtmb_single
1| ON| 0| OFF

set_source(*dtmb_source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:SOURce
driver.source.bb.dtmb.set_source(dtmb_source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param dtmb_source
EXTernal| TSPLayer| TESTsignal

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:STATe
driver.source.bb.dtmb.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

set_stuffing(*dtmb_stuffing: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:STUFfing
driver.source.bb.dtmb.set_stuffing(dtmb_stuffing = False)
```

Activates stuffing.

param dtmb_stuffing
1| ON| 0| OFF

set_test_signal(*dtmb_test_sig: CodingInputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:TESTsignal
driver.source.bb.dtmb.set_test_signal(dtmb_test_sig = enums.
↪CodingInputSignalTestSignal.TTSP)
```

Queries the test signal, that consists of test packets.

param dtmb_test_sig
TTSP Test TS packet with standardized packet data used as modulation data in the transport stream.

set_ts_packet(*dtmb_test_ts_pack: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:TSPacket
driver.source.bb.dtmb.set_ts_packet(dtmb_test_ts_pack = enums.
↪SettingsTestTsPacket.H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param dtmb_test_ts_pack
H184| S187

Cloning the Group

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

Subgroups

6.18.3.7.1 Channel

SCPI Command :

```
[SOURce<HW>]:BB:DTMB:CHANnel:[BANDwidth]
```

class ChannelCls

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

get_bandwidth() → CodingChannelBandwidth

```
# SCPI: [SOURce<HW>]:BB:DTMB:CHANnel:[BANDwidth]
value: enums.CodingChannelBandwidth = driver.source.bb.dtmb.channel.get_
↪bandwidth()
```

Selects the channel bandwidth.

return
dtmb_bandwidth: BW_6| BW_7| BW_8

set_bandwidth(*dtmb_bandwidth: CodingChannelBandwidth*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:CHANnel:[BANDwidth]
driver.source.bb.dtmb.channel.set_bandwidth(dtmb_bandwidth = enums.
↪CodingChannelBandwidth.BW_6)
```

Selects the channel bandwidth.

param dtmb_bandwidth
BW_6| BW_7| BW_8

6.18.3.7.2 Dual

SCPI Command :

```
[SOURce<HW>]:BB:DTMB:DUAL:PILot
```

class DualCls

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

get_pilot() → bool

```
# SCPI: [SOURce<HW>]:BB:DTMB:DUAL:PILot
value: bool = driver.source.bb.dtmb.dual.get_pilot()
```

Enables/disables insertion of the dual pilot tone.

return
dtmb_dual_pilot: 1| ON| 0| OFF

set_pilot(*dtmb_dual_pilot: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:DUAL:PILot
driver.source.bb.dtmb.dual.set_pilot(dtmb_dual_pilot = False)
```

Enables/disables insertion of the dual pilot tone.

param dtmb_dual_pilot
1| ON| 0| OFF

6.18.3.7.3 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:DTMB:INPut:FORMat
[SOURce<HW>]:BB:DTMB:INPut:TSCHannel
[SOURce<HW>]:BB:DTMB:[INPut]:DATarate
[SOURce<HW>]:BB:DTMB:INPut
```

class InputPyCls

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

get_data_rate() → float

```
# SCPI: [SOURce<HW>]:BB:DTMB:[INPut]:DATarate
value: float = driver.source.bb.dtmb.inputPy.get_data_rate()


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
```

(continues on next page)

(continued from previous page)

```
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
dtmb_meas: float Range: 0 to 999999999

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:DTMB:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.dtmb.inputPy.get_format_py()
```

Sets the format of the input signal.

return
dtmb_format: ASI| SMPTE

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:DTMB:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.dtmb.inputPy.get_ts_channel()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

INTRO_CMD_HELP: For configuring IP channel settings and local IP data network parameters, see:

- 'IP subsystem'
- 'BCIP subsystem'

return
dtmb_ip_ts_channel: 1| 2| 3| 4

get_value() → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:DTMB:INPut
value: enums.CodingInputSignalInputA = driver.source.bb.dtmb.inputPy.get_value()
```

Sets the external input interface.

return
dtmb_source: TS| IP

set_format_py(*dtmb_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:INPut:FORMat
driver.source.bb.dtmb.inputPy.set_format_py(dtmb_format = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param dtmb_format
ASI| SMPTE

set_ts_channel(*dtmb_ip_ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:INPut:TSCHannel
driver.source.bb.dtmb.inputPy.set_ts_channel(dtmb_ip_ts_channel = enums.NumberA.
↪_1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

INTRO_CMD_HELP: For configuring IP channel settings and local IP data network parameters, see:

- 'IP subsystem'
- 'BCIP subsystem'

param dtmb_ip_ts_channel
1| 2| 3| 4

set_value(*dtmb_source: CodingInputSignalInputA*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:INPut
driver.source.bb.dtmb.inputPy.set_value(dtmb_source = enums.
↪CodingInputSignalInputA.ASI1)
```

Sets the external input interface.

param dtmb_source
TS| IP

6.18.3.7.4 Prbs

SCPI Command :

```
[SOURce<HW>]:BB:DTMB:PRBS:[SEQuence]
```

class PrbsCls

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

get_sequence() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:DTMB:PRBS:[SEQuence]
value: enums.SettingsPrbs = driver.source.bb.dtmb.prbs.get_sequence()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
dtmb_prbs: P23_1| P15_1

set_sequence(*dtmb_prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:PRBS:[SEQuence]
driver.source.bb.dtmb.prbs.set_sequence(dtmb_prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param dtmb_prbs
P23_1| P15_1

6.18.3.7.5 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:DTMB:SETTing:CATalog
[SOURce<HW>]:BB:DTMB:SETTing:DELete
[SOURce<HW>]:BB:DTMB:SETTing:LOAD
[SOURce<HW>]:BB:DTMB:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:SETTing:DELete
driver.source.bb.dtmb.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.dtmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DTMB:SETTing:CATalog
value: List[str] = driver.source.bb.dtmb.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.dtmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
catalog: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:DTMB:SETTing:LOAD
value: str = driver.source.bb.dtmb.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dtmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: string

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:DTMB:SETTing:STORe
value: str = driver.source.bb.dtmb.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.dtmb) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: string

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:SETTing:LOAD
driver.source.bb.dtmb.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dtmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
string

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:SETTing:STORe
driver.source.bb.dtmb.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.dtmb) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
string

6.18.3.7.6 Special

SCPI Command :

```
[SOURce<HW>]:BB:DTMB:[SPECial]:SIPNormal
```

class SpecialCls

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

get_sip_normal() → bool

```
# SCPI: [SOURce<HW>]:BB:DTMB:[SPECial]:SIPNormal
value: bool = driver.source.bb.dtmb.special.get_sip_normal()
```

Enables or disables the system information (SI) power normalization.

return
dtmb_sip_normal: 1| ON| 0| OFF

set_sip_normal(*dtmb_sip_normal: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:[SPECial]:SIPNormal
driver.source.bb.dtmb.special.set_sip_normal(dtmb_sip_normal = False)
```

Enables or disables the system information (SI) power normalization.

param dtmb_sip_normal
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dtmb.special.clone()
```

Subgroups

6.18.3.7.6.1 Settings

SCPI Command :

```
[SOURce<HW>]:BB:DTMB:[SPECial]:SETTings:[STATe]
```

class SettingsCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DTMB:[SPECial]:SETTings:[STATe]
value: bool = driver.source.bb.dtmb.special.settings.get_state()
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

return
dtmb_special: 1| ON| 0| OFF

set_state(*dtmb_special: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:[SPECial]:SETTings:[STATe]
driver.source.bb.dtmb.special.settings.set_state(dtmb_special = False)
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

param dtmb_special
1| ON| 0| OFF

6.18.3.7.7 Time

SCPI Command :

```
[SOURce<HW>]:BB:DTMB:TIME:[INTerleaver]
```

class TimeCls

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

get_interleaver() → DtmbCodingTimeInterleaver

```
# SCPI: [SOURce<HW>]:BB:DTMB:TIME:[INTerleaver]
value: enums.DtmbCodingTimeInterleaver = driver.source.bb.dtmb.time.get_
↪interleaver()
```

Defines the depth of the basic delay.

return
dtmb_time_int: OFF| I240| I720 I240|I720 Basic delay of 240/720 symbols OFF Disables/bridges the time interleaver.

set_interleaver(*dtmb_time_int: DtmbCodingTimeInterleaver*) → None

```
# SCPI: [SOURce<HW>]:BB:DTMB:TIME:[INTerleaver]
driver.source.bb.dtmb.time.set_interleaver(dtmb_time_int = enums.
↪DtmbCodingTimeInterleaver.I240)
```

Defines the depth of the basic delay.

param dtmb_time_int
OFF| I240| I720 I240|I720 Basic delay of 240/720 symbols OFF Disables/bridges the time interleaver.

6.18.3.7.8 Useful

class UsefulCls

Useful 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.source.bb.dtmb.useful.clone()
```

Subgroups

6.18.3.7.8.1 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:DTMB:USEFul:[RATE]:MAX
[SOURce<HW>]:BB:DTMB:USEFul:[RATE]
```

class RateCls

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

get_max() → int

```
# SCPI: [SOURce<HW>]:BB:DTMB:USEFul:[RATE]:MAX
value: int = driver.source.bb.dtmb.useful.rate.get_max()
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

return
dtmb_max_use: integer Range: 0 to 999999999

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:DTMB:USEFul:[RATE]
value: float = driver.source.bb.dtmb.useful.rate.get_value()
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

return
dtmb_useful: float Range: 0 to 999999999

6.18.3.8 Dvbc

SCPI Commands :

```
[SOURce<HW>]:BB:DVBC:CONStel
[SOURce<HW>]:BB:DVBC:PACKetlength
[SOURce<HW>]:BB:DVBC:PAYLoad
[SOURce<HW>]:BB:DVBC:PID
[SOURce<HW>]:BB:DVBC:PIDTestpack
[SOURce<HW>]:BB:DVBC:PRBS
[SOURce<HW>]:BB:DVBC:PRESet
[SOURce<HW>]:BB:DVBC:ROLLoff
[SOURce<HW>]:BB:DVBC:SOURce
[SOURce<HW>]:BB:DVBC:STATe
[SOURce<HW>]:BB:DVBC:STUFfing
[SOURce<HW>]:BB:DVBC:SYMBols
[SOURce<HW>]:BB:DVBC:TESTsignal
[SOURce<HW>]:BB:DVBC:TSPacket
```

class DvbcCls

Dvbc commands group definition. 26 total commands, 4 Subgroups, 14 group commands

get_constel() → DvbcCodingDvbcCodingConstel

```
# SCPI: [SOURce<HW>]:BB:DVBC:CONStel
value: enums.DvbcCodingDvbcCodingConstel = driver.source.bb.dvbc.get_constel()
```

Defines the constellation.

return
constel: C16| C32| C64| C128| C256 C16|C32|C64|C128|C256 16/32/64/128/256QAM

get_packet_length() → DvbxCodingInputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:DVBC:PACKetlength
value: enums.DvbxCodingInputSignalPacketLength = driver.source.bb.dvbc.get_
↪packet_length()
```

Queries the packet length of the external transport stream in bytes.

return
inp_sig_plength: P188| P204| INValid P188|P204 188/204 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:DVBC:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.dvbc.get_payload()
```

Defines the payload area content of the packet.

return
set_payload: PRBS| H00| HFF PRBS PRBS data in accordance with H00 Exclusively 00 (hex) data HFF Exclusively FF (hex) data

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:DVBC:PID
value: int = driver.source.bb.dvbc.get_pid()
```

Sets the .

return
set_pid: integer Range: 0 to 8191

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:DVBC:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.dvbc.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
set_pid_testpack: NULL| VARiable

get_prbs() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:DVBC:PRBS
value: enums.SettingsPrbs = driver.source.bb.dvbc.get_prbs()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
set_prbs: P23_1| P15_1 P23_1 PRBS 23 sequence as specified by . P15_1 PRBS 15 sequence as specified by .

get_rolloff() → DvbcCodingDvbcCodingRolloff

```
# SCPI: [SOURce<HW>]:BB:DVBC:ROLLoff
value: enums.DvbcCodingDvbcCodingRolloff = driver.source.bb.dvbc.get_rolloff()
```

Displays the roll-off factor.

return
rolloff: 0.13| 0.15

get_source() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DVBC:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.dvbc.get_source()
```

Sets the modulation source for the input signal.

return
inp_sig_source: EXTernal| TSPLayer| TESTsignal

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBC:STATe
value: bool = driver.source.bb.dvbc.get_state()
```

Enables/disables the DVB-S standard.

return
state: 1| ON| 0| OFF

get_stuffing() → StateOn

```
# SCPI: [SOURce<HW>]:BB:DVBC:STUFfing
value: enums.StateOn = driver.source.bb.dvbc.get_stuffing()
```

Queries the stuffing state that is active.

return
inp_sig_stuffing: 1| ON

get_symbols() → int

```
# SCPI: [SOURce<HW>]:BB:DVBC:SYMBols
value: int = driver.source.bb.dvbc.get_symbols()
```

Sets the symbol rate.

return
symbol_rate: integer Range: 1.00E+05 to 8.00E+07

get_test_signal() → DvbcCodingDvbcInputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:DVBC:TESTsignal
value: enums.DvbcCodingDvbcInputSignalTestSignal = driver.source.bb.dvbc.get_
↪test_signal()
```

Defines the test signal data.

return
inp_sig_test_sig: TTSP| PBDE| PBEM TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBDE PRBS before differential encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the differential encoder. PRBS data conforms with specification. PBEM PRBS before mapper Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the mapper.

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:DVBC:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.dvbc.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
set_ts_packet: H184| S187 H184 Head/184 Payload S187 Sync/187 Payload

preset() → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:PRESet
driver.source.bb.dvbc.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>:BB:DVBC:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:DVBC:PRESet
driver.source.bb.dvbc.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>:BB:DVBC:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_constel(*constel: DvbcCodingDvbcCodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:CONStel
driver.source.bb.dvbc.set_constel(constel = enums.DvbcCodingDvbcCodingConstel.
↪C128)
```

Defines the constellation.

param constel
C16| C32| C64| C128| C256 C16|C32|C64|C128|C256 16/32/64/128/256QAM

set_payload(*set_payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:PAYLoad
driver.source.bb.dvbc.set_payload(set_payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param set_payload
PRBS| H00| HFF PRBS PRBS data in accordance with H00 Exclusively 00 (hex) data HFF Exclusively FF (hex) data

set_pid(*set_pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:PID
driver.source.bb.dvbc.set_pid(set_pid = 1)
```

Sets the .

param set_pid
integer Range: 0 to 8191

set_pid_test_pack(*set_pid_testpack: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:PIDTestpack
driver.source.bb.dvbc.set_pid_test_pack(set_pid_testpack = enums.PidTestPacket.
↪NULL)
```

If a header is present in the test packet ('Test TS Packet > Head/184 Payload') , you can specify a fixed or variable packet identifier (PID) .

param set_pid_testpack
NULL| VARiable

set_prbs(*set_prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:PRBS
driver.source.bb.dvbc.set_prbs(set_prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param set_prbs
P23_1| P15_1 P23_1 PRBS 23 sequence as specified by . P15_1 PRBS 15 sequence as specified by .

set_rolloff(*rolloff: DvbcCodingDvbcCodingRolloff*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:ROLLoff
driver.source.bb.dvbc.set_rolloff(rolloff = enums.DvbcCodingDvbcCodingRolloff._
↪0_dot_13)
```

Displays the roll-off factor.

param rolloff
0.13| 0.15

set_source(*inp_sig_source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:SOURce
driver.source.bb.dvbc.set_source(inp_sig_source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param inp_sig_source
EXTernal| TSPLayer| TESTsignal

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:STATe
driver.source.bb.dvbc.set_state(state = False)
```

Enables/disables the DVB-S standard.

param state
1| ON| 0| OFF

set_symbols(*symbol_rate: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:SYMBols
driver.source.bb.dvbc.set_symbols(symbol_rate = 1)
```

Sets the symbol rate.

param symbol_rate
integer Range: 1.00E+05 to 8.00E+07

set_test_signal(*inp_sig_test_sig: DvbcCodingDvbcInputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:TESTsignal
driver.source.bb.dvbc.set_test_signal(inp_sig_test_sig = enums.
↪DvbcCodingDvbcInputSignalTestSignal.PBDE)
```

Defines the test signal data.

param inp_sig_test_sig
TTSP| PBDE| PBEM TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBDE PRBS before differential encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the differential encoder. PRBS data conforms with specification. PBEM PRBS before mapper Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the mapper.

set_ts_packet(*set_ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:TSPacket
driver.source.bb.dvbc.set_ts_packet(set_ts_packet = enums.SettingsTestTsPacket.
↪H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param set_ts_packet
H184| S187 H184 Head/184 Payload S187 Sync/187 Payload

Cloning the Group

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

Subgroups

6.18.3.8.1 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:DVBC:INPut:FORMat
[SOURce<HW>]:BB:DVBC:INPut:TSCHannel
[SOURce<HW>]:BB:DVBC:[INPut]:DATarate
[SOURce<HW>]:BB:DVBC:INPut
```

class InputPyCls

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

get_data_rate() → float

```
# SCPI: [SOURce<HW>]:BB:DVBC:[INPut]:DATarate
value: float = driver.source.bb.dvbc.inputPy.get_data_rate()
```

Queries the measured value of the data rate of one of the following: External transport stream including null packets input at 'User 1' connector External transport stream including null packets input at 'IP Data/LAN' connector (TSoverIP) The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
inp_sig_datarate: float Range: 0 to 999999999

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:DVBC:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.dvbc.inputPy.get_format_py()
```

Sets the format of the input signal.

return
inp_sig_format: ASI| SMPTE

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:DVBC:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.dvbc.inputPy.get_ts_channel()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

return
inp_sig_ts_channel: 1| 2| 3| 4

get_value() → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:DVBC:INPut
value: enums.CodingInputSignalInputA = driver.source.bb.dvbc.inputPy.get_value()
```

Sets the external input interface.

return
inp_sig_input: TS| IP TS Input for serial transport stream data. The signal is input at the 'User 1/2' connectors. IP Input for IP transport stream data. The signal is input at the 'IP Data' connector.

set_format_py(*inp_sig_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:INPut:FORMat
driver.source.bb.dvbc.inputPy.set_format_py(inp_sig_format = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param inp_sig_format
ASI| SMPTE

set_ts_channel(*inp_sig_ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:INPut:TSCHannel
driver.source.bb.dvbc.inputPy.set_ts_channel(inp_sig_ts_channel = enums.NumberA.
↪_1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param inp_sig_ts_channel
1| 2| 3| 4

set_value(*inp_sig_input: CodingInputSignalInputA*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:INPut
driver.source.bb.dvbc.inputPy.set_value(inp_sig_input = enums.
↪CodingInputSignalInputA.ASI1)
```

Sets the external input interface.

param inp_sig_input
TS| IP TS Input for serial transport stream data. The signal is input at the 'User 1/2' connectors. IP Input for IP transport stream data. The signal is input at the 'IP Data' connector.

6.18.3.8.2 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:DVBC:SETTing:CATalog
[SOURce<HW>]:BB:DVBC:SETTing:DELete
[SOURce<HW>]:BB:DVBC:SETTing:LOAD
[SOURce<HW>]:BB:DVBC:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:SETTing:DELete
driver.source.bb.dvbc.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.dvbc. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted.

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DVBC:SETTing:CATalog
value: List[str] = driver.source.bb.dvbc.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.dvbc. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
catalog: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:DVBC:SETTing:LOAD
value: str = driver.source.bb.dvbc.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dvbc. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: 'filename' Filename or complete file path; file extension can be omitted.

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:DVBC:SETTing:STORe
value: str = driver.source.bb.dvbc.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.dvbc) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: 'filename' Filename or complete file path

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:SETTing:LOAD
driver.source.bb.dvbc.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dvbc. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
'filename' Filename or complete file path; file extension can be omitted.

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:SETTing:STORe
driver.source.bb.dvbc.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.dvbc) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
'filename' Filename or complete file path

6.18.3.8.3 Special

SCPI Command :

```
[SOURce<HW>]:BB:DVBC:[SPECial]:REEDsolomon
```

class SpecialCls

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

get_reed_solomon() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBC:[SPECial]:REEDsolomon
value: bool = driver.source.bb.dvbc.special.get_reed_solomon()
```

Enables/disables the Reed-Solomon encoder.

return
reed_solomon: 1| ON| 0| OFF

set_reed_solomon(*reed_solomon: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:[SPECial]:REEDsolomon
driver.source.bb.dvbc.special.set_reed_solomon(reed_solomon = False)
```

Enables/disables the Reed-Solomon encoder.

param reed_solomon
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbc.special.clone()
```

Subgroups

6.18.3.8.3.1 Setting

SCPI Command :

```
[SOURce<HW>]:BB:DVBC:[SPECial]:SETTing:[STATe]
```

class SettingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBC:[SPECial]:SETTing:[STATe]
value: bool = driver.source.bb.dvbc.special.setting.get_state()
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

return
special_state: 1| ON| 0| OFF

set_state(*special_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBC:[SPECial]:SETTing:[STATe]
driver.source.bb.dvbc.special.setting.set_state(special_state = False)
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

param special_state
1| ON| 0| OFF

6.18.3.8.4 Useful

class UsefulCls

Useful 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.source.bb.dvbc.useful.clone()
```

Subgroups

6.18.3.8.4.1 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:DVBC:USEFul:[RATE]:MAX
[SOURce<HW>]:BB:DVBC:USEFul:[RATE]
```

class RateCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:BB:DVBC:USEFul:[RATE]:MAX
value: float = driver.source.bb.dvbc.useful.rate.get_max()
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

return
inp_sig_max_rate: float Range: 0 to 999999999

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:DVBC:USEFul:[RATE]
value: float = driver.source.bb.dvbc.useful.rate.get_value()
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

return
inp_sig_usefull: float Range: 0 to 999999999

6.18.3.9 Dvbs

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS:CONStel
[SOURce<HW>]:BB:DVBS:PACKetlength
[SOURce<HW>]:BB:DVBS:PAYLoad
[SOURce<HW>]:BB:DVBS:PID
[SOURce<HW>]:BB:DVBS:PIDTestpack
[SOURce<HW>]:BB:DVBS:PRBS
[SOURce<HW>]:BB:DVBS:PRESet
[SOURce<HW>]:BB:DVBS:RATE
[SOURce<HW>]:BB:DVBS:ROLLoff
```

(continues on next page)

(continued from previous page)

```
[SOURce<HW>]:BB:DVBS:SOURce
[SOURce<HW>]:BB:DVBS:STATe
[SOURce<HW>]:BB:DVBS:STUFfing
[SOURce<HW>]:BB:DVBS:SYMBols
[SOURce<HW>]:BB:DVBS:TESTsignal
[SOURce<HW>]:BB:DVBS:TSPacket
```

class DvbsCls

Dvbs commands group definition. 27 total commands, 4 Subgroups, 15 group commands

get_constel() → DvbsCodingDvbsCodingConstel

```
# SCPI: [SOURce<HW>]:BB:DVBS:CONStel
value: enums.DvbsCodingDvbsCodingConstel = driver.source.bb.dvbs.get_constel()
```

Defines the constellation.

return
constel: S4| S8| S16 S4 S8 8 S16 16

get_packet_length() → DvbxCodingInputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:DVBS:PACKetlength
value: enums.DvbxCodingInputSignalPacketLength = driver.source.bb.dvbs.get_
↪packet_length()
```

Queries the packet length of the external transport stream in bytes.

return
inp_sig_plength: P188| P204| INValid P188|P204 188/204 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:DVBS:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.dvbs.get_payload()
```

Defines the payload area content of the packet.

return
set_payload: No help available

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:DVBS:PID
value: int = driver.source.bb.dvbs.get_pid()
```

Sets the .

return
set_pid: No help available

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:DVBS:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.dvbs.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
set_pid_testpack: No help available

get_prbs() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:DVBS:PRBS
value: enums.SettingsPrbs = driver.source.bb.dvbs.get_prbs()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
set_prbs: P23_1| P15_1

get_rate() → DvbsCodingDvbsCodingCoderate

```
# SCPI: [SOURce<HW>]:BB:DVBS:RATE
value: enums.DvbsCodingDvbsCodingCoderate = driver.source.bb.dvbs.get_rate()
```

Defines the code rate. The available code rates depend on the value of [:SOURce<hw>]:BB:DVBS:CONStel.

return
coderate: R1_2| R2_3| R3_4| R5_6| R7_8| R8_9

get_rolloff() → DvbsCodingDvbsCodingRolloff

```
# SCPI: [SOURce<HW>]:BB:DVBS:ROLLoff
value: enums.DvbsCodingDvbsCodingRolloff = driver.source.bb.dvbs.get_rolloff()
```

Sets the roll-off alpha factor value.

return
rolloff: 0.35| 0.25| 0.20| 0.15| 0.10| 0.05

get_source() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DVBS:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.dvbs.get_source()
```

Sets the modulation source for the input signal.

return
inp_sig_source: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS:STATe
value: bool = driver.source.bb.dvbs.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_stuffing() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS:STUFfing
value: bool = driver.source.bb.dvbs.get_stuffing()
```

Queries the stuffing state that is active.

return
inp_sig_stuffing: 1| ON| 0| OFF

get_symbols() → int

```
# SCPI: [SOURce<HW>]:BB:DVBS:SYMBols
value: int = driver.source.bb.dvbs.get_symbols()
```

Sets the symbol rate.

return
symbol_rate: integer Range: 1.00E+05 to 9.00E+07

get_test_signal() → DvbsCodingDvbsInputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:DVBS:TESTsignal
value: enums.DvbsCodingDvbsInputSignalTestSignal = driver.source.bb.dvbs.get_
↪test_signal()
```

Defines the test signal data.

return
inp_sig_test_sig: TTSP| PBEC TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBEC PRBS before convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the convolutional encoder. PRBS data conforms with specification.

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:DVBS:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.dvbs.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
set_ts_packet: No help available

preset() → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:PRESet
driver.source.bb.dvbs.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>:BB:DVBS2:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:DVBS:PRESet
driver.source.bb.dvbs.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>:BB:DVBS2:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_constel(*constel: DvbsCodingDvbsCodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:CONStel
driver.source.bb.dvbs.set_constel(constel = enums.DvbsCodingDvbsCodingConstel.
↪S16)
```

Defines the constellation.

param constel
S4| S8| S16 S4 S8 8 S16 16

set_payload(*set_payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:PAYLoad
driver.source.bb.dvbs.set_payload(set_payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param set_payload
HFF| H00| PRBS

set_pid(*set_pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:PID
driver.source.bb.dvbs.set_pid(set_pid = 1)
```

Sets the .

param set_pid
float Range: #H0 to #HFFF

set_pid_test_pack(*set_pid_testpack: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:PIDTestpack
driver.source.bb.dvbs.set_pid_test_pack(set_pid_testpack = enums.PidTestPacket.
↪NULL)
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

param set_pid_testpack
VARiable| NULL

set_prbs(*set_prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:PRBS
driver.source.bb.dvbs.set_prbs(set_prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param set_prbs
P23_1| P15_1

set_rate(*coderate: DvbsCodingDvbsCodingCoderate*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:RATE
driver.source.bb.dvbs.set_rate(coderate = enums.DvbsCodingDvbsCodingCoderate.R1_
↪2)
```

Defines the code rate. The available code rates depend on the value of [:SOURce<hw>]:BB:DVBS:CONStel.

param coderate
R1_2| R2_3| R3_4| R5_6| R7_8| R8_9

set_rolloff(*rolloff: DvbsCodingDvbsCodingRolloff*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:ROLLoff
driver.source.bb.dvbs.set_rolloff(rolloff = enums.DvbsCodingDvbsCodingRolloff._
↪0_dot_20)
```

Sets the roll-off alpha factor value.

param rolloff
0.35| 0.25| 0.20| 0.15| 0.10| 0.05

set_source(*inp_sig_source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:SOURce
driver.source.bb.dvbs.set_source(inp_sig_source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param inp_sig_source
EXTernal| TSPLayer| TESTsignal

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:STATe
driver.source.bb.dvbs.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

set_symbols(*symbol_rate: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:SYMBols
driver.source.bb.dvbs.set_symbols(symbol_rate = 1)
```

Sets the symbol rate.

param symbol_rate
integer Range: 1.00E+05 to 9.00E+07

set_test_signal(*inp_sig_test_sig: DvbsCodingDvbsInputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:TESTsignal
driver.source.bb.dvbs.set_test_signal(inp_sig_test_sig = enums.
↪DvbsCodingDvbsInputSignalTestSignal.PBEC)
```

Defines the test signal data.

param inp_sig_test_sig
TTSP| PBEC TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBEC PRBS before convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the convolutional encoder. PRBS data conforms with specification.

set_ts_packet(*set_ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:TSPacket
driver.source.bb.dvbs.set_ts_packet(set_ts_packet = enums.SettingsTestTsPacket.
↪H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param set_ts_packet
H184| S187

Cloning the Group

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

Subgroups

6.18.3.9.1 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS:INPut:FORMat
[SOURce<HW>]:BB:DVBS:INPut:TSCHannel
[SOURce<HW>]:BB:DVBS:[INPut]:DATarate
[SOURce<HW>]:BB:DVBS:INPut
```

class InputPyCls

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

get_data_rate() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS:[INPut]:DATarate
value: float = driver.source.bb.dvbs.inputPy.get_data_rate()


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
```

(continues on next page)

(continued from previous page)

```
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
inp_sig_datarate: float Range: 0 to 999999999

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:DVBS:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.dvbs.inputPy.get_format_py()
```

Sets the format of the input signal.

return
inp_sig_format: ASI| SMPTE

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:DVBS:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.dvbs.inputPy.get_ts_channel()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

INTRO_CMD_HELP: For configuring IP channel settings and local IP data network parameters, see:

- 'IP subsystem'
- 'BCIP subsystem'

return
inp_sig_ts_channel: 1| 2| 3| 4

get_value() → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:DVBS:INPut
value: enums.CodingInputSignalInputA = driver.source.bb.dvbs.inputPy.get_value()
```

Sets the external input interface.

return
inp_sig_input: TS| IP

set_format_py(*inp_sig_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:INPut:FORMat
driver.source.bb.dvbs.inputPy.set_format_py(inp_sig_format = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param inp_sig_format
ASI| SMPTE

set_ts_channel(*inp_sig_ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:INPut:TSCHannel
driver.source.bb.dvbs.inputPy.set_ts_channel(inp_sig_ts_channel = enums.NumberA.
↪_1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

INTRO_CMD_HELP: For configuring IP channel settings and local IP data network parameters, see:

- 'IP subsystem'
- 'BCIP subsystem'

param inp_sig_ts_channel
1| 2| 3| 4

set_value(*inp_sig_input: CodingInputSignalInputA*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:INPut
driver.source.bb.dvbs.inputPy.set_value(inp_sig_input = enums.
↪CodingInputSignalInputA.ASI1)
```

Sets the external input interface.

param inp_sig_input
TS| IP

6.18.3.9.2 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS:SETTing:CATalog
[SOURce<HW>]:BB:DVBS:SETTing:DELete
[SOURce<HW>]:BB:DVBS:SETTing:LOAD
[SOURce<HW>]:BB:DVBS:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:SETTing:DELete
driver.source.bb.dvbs.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DVBS:SETTing:CATalog
value: List[str] = driver.source.bb.dvbs.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
catalog: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:DVBS:SETTing:LOAD
value: str = driver.source.bb.dvbs.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: No help available

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:DVBS:SETTing:STORe
value: str = driver.source.bb.dvbs.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.d2vbs) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: 'filename' Filename or complete file path

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:SETTing:LOAD
driver.source.bb.dvbs.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
'filename' Filename or complete file path; file extension can be omitted.

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:SETTing:STORe
driver.source.bb.dvbs.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.d2vbs) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
'filename' Filename or complete file path

6.18.3.9.3 Special

SCPI Command :

```
[SOURce<HW>]:BB:DVBS:[SPECial]:REEDsolomon
```

class SpecialCls

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

get_reed_solomon() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS:[SPECial]:REEDsolomon
value: bool = driver.source.bb.dvbs.special.get_reed_solomon()
```

Enables the Reed-Solomon encoder. The standard stipulates a Reed-Solomon RS (204, 188) .

return
reed_solomon: 1| ON| 0| OFF

set_reed_solomon(*reed_solomon: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:[SPECial]:REEDsolomon
driver.source.bb.dvbs.special.set_reed_solomon(reed_solomon = False)
```

Enables the Reed-Solomon encoder. The standard stipulates a Reed-Solomon RS (204, 188) .

param reed_solomon
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs.special.clone()
```

Subgroups

6.18.3.9.3.1 Setting

SCPI Command :

```
[SOURce<HW>]:BB:DVBS:[SPECial]:SETTing:[STATe]
```

class SettingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS:[SPECial]:SETTing:[STATe]
value: bool = driver.source.bb.dvbs.special.setting.get_state()
```

Enables or disables all special settings.

return
special_state: No help available

set_state(*special_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS:[SPECial]:SETTing:[STATe]
driver.source.bb.dvbs.special.setting.set_state(special_state = False)
```

Enables or disables all special settings.

param special_state
1| ON| 0| OFF

6.18.3.9.4 Useful

class UsefulCls

Useful 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.source.bb.dvbs.useful.clone()
```

Subgroups

6.18.3.9.4.1 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS:USEFul:[RATE]:MAX
[SOURce<HW>]:BB:DVBS:USEFul:[RATE]
```

class RateCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS:USEFul:[RATE]:MAX
value: float = driver.source.bb.dvbs.useful.rate.get_max()
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

return
inp_sig_max_rate: float Range: 0 to 999999999

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS:USEFul:[RATE]
value: float = driver.source.bb.dvbs.useful.rate.get_value()
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

return
inp_sig_usefull: float Range: 0 to 999999999

6.18.3.10 Dvbs2

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS2:ANNM
[SOURce<HW>]:BB:DVBS2:NTSL
[SOURce<HW>]:BB:DVBS2:PAYLoad
[SOURce<HW>]:BB:DVBS2:PID
[SOURce<HW>]:BB:DVBS2:PIDTestpack
[SOURce<HW>]:BB:DVBS2:PRESet
[SOURce<HW>]:BB:DVBS2:ROLLoff
[SOURce<HW>]:BB:DVBS2:STATe
[SOURce<HW>]:BB:DVBS2:TSPacket
```

class Dvbs2Cls

Dvbs2 commands group definition. 42 total commands, 9 Subgroups, 9 group commands

get_annm() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:ANNM
value: bool = driver.source.bb.dvbs2.get_annm()
```

Enables the annex M features as specified in . Depending on this setting, a different PL header is used.

return
annex_n: 1| ON| 0| OFF

get_ntsl() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:NTSL
value: float = driver.source.bb.dvbs2.get_ntsl()
```

No command help available

return
num_time_slice: float Range: 1 to 8

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.dvbs2.get_payload()
```

Defines the payload area content of the packet.

return
payload: HFF| H00| PRBS

get_pid() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PID
value: float = driver.source.bb.dvbs2.get_pid()
```

Sets the .

return
pid: float Range: #H0 to #HFFF

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.dvbs2.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
pid_test_packet: VARiable| NULL

get_rolloff() → Dvbs2CodingRolloff

```
# SCPI: [SOURce<HW>]:BB:DVBS2:ROLLoff
value: enums.Dvbs2CodingRolloff = driver.source.bb.dvbs2.get_rolloff()
```

Sets the roll-off alpha factor value.

return
rolloff: 0.35| 0.25| 0.20| 0.15| 0.10| 0.05

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:STATe
value: bool = driver.source.bb.dvbs2.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.dvbs2.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
ts_packet: H184| S187

preset() → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PRESet
driver.source.bb.dvbs2.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>:BB:DVBS2:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PRESet
driver.source.bb.dvbs2.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>:BB:DVBS2:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_annm(*annex_n: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:ANNM
driver.source.bb.dvbs2.set_annm(annex_n = False)
```

Enables the annex M features as specified in . Depending on this setting, a different PL header is used.

param annex_n
1| ON| 0| OFF

set_ntsl(*num_time_slice: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:NTSL
driver.source.bb.dvbs2.set_ntsl(num_time_slice = 1.0)
```

No command help available

param num_time_slice
float Range: 1 to 8

set_payload(*payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PAYLoad
driver.source.bb.dvbs2.set_payload(payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param payload
HFF| H00| PRBS

set_pid(*pid: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PID
driver.source.bb.dvbs2.set_pid(pid = 1.0)
```

Sets the .

param pid
float Range: #H0 to #HFFF

set_pid_test_pack(*pid_test_packet: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PIDTestpack
driver.source.bb.dvbs2.set_pid_test_pack(pid_test_packet = enums.PidTestPacket.
↪NULL)
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

param pid_test_packet
VARiable| NULL

set_rolloff(*rolloff: Dvbs2CodingRolloff*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:ROLLoff
driver.source.bb.dvbs2.set_rolloff(rolloff = enums.Dvbs2CodingRolloff._0_dot_05)
```

Sets the roll-off alpha factor value.

param rolloff
0.35| 0.25| 0.20| 0.15| 0.10| 0.05

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:STATe
driver.source.bb.dvbs2.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

set_ts_packet(*ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSPacket
driver.source.bb.dvbs2.set_ts_packet(ts_packet = enums.SettingsTestTsPacket.
↪H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param ts_packet
H184| S187

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.clone()
```

Subgroups

6.18.3.10.1 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS2:[INPut]:CMMode
[SOURce<HW>]:BB:DVBS2:[INPut]:NIS
```

class InputPyCls

InputPy commands group definition. 6 total commands, 1 Subgroups, 2 group commands

get_cm_mode() → Dvbs2InputSignalCmMode

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[INPut]:CMMode
value: enums.Dvbs2InputSignalCmMode = driver.source.bb.dvbs2.inputPy.get_cm_
↪mode()
```

Sets the coding and modulation (CM) mode.

return
cm_mode: VCM| CCM| ACM

get_nis() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[INPut]:NIS
value: float = driver.source.bb.dvbs2.inputPy.get_nis()
```

Sets the number of input streams. Maximum 8 input streams are possible.

return
num_inp_sig: float Range: 1 to 8

set_cm_mode(*cm_mode: Dvbs2InputSignalCmMode*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[INPut]:CMMode
driver.source.bb.dvbs2.inputPy.set_cm_mode(cm_mode = enums.
↪Dvbs2InputSignalCmMode.ACM)
```

Sets the coding and modulation (CM) mode.

param cm_mode
VCM| CCM| ACM

set_nis(*num_inp_sig: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[INPut]:NIS
driver.source.bb.dvbs2.inputPy.set_nis(num_inp_sig = 1.0)
```

Sets the number of input streams. Maximum 8 input streams are possible.

param num_inp_sig
float Range: 1 to 8

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.inputPy.clone()
```

Subgroups

6.18.3.10.1.1 IsPy<InputStream>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.bb.dvbs2.inputPy.isPy.repcap_inputStream_get()
driver.source.bb.dvbs2.inputPy.isPy.repcap_inputStream_set(repcap.InputStream.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]
```

class IsPyCls

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

get(*inputStream=InputStream.Default*) → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]
value: enums.CodingInputSignalInputA = driver.source.bb.dvbs2.inputPy.isPy.
↪get(inputStream = repcap.InputStream.Default)
```

Sets the external input interface.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
input_py: TS| IP

set(*input_py: CodingInputSignalInputA*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]
driver.source.bb.dvbs2.inputPy.isPy.set(input_py = enums.
↪CodingInputSignalInputA.ASI1, inputStream = repcap.InputStream.Default)
```

Sets the external input interface.

param input_py
TS| IP

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.inputPy.isPy.clone()
```

Subgroups

6.18.3.10.1.2 DataRate

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[INPut]:[IS<CH>]:DATarate
```

class DataRateCls

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

get(*inputStream=InputStream.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[INPut]:[IS<CH>]:DATarate
value: float = driver.source.bb.dvbs2.inputPy.isPy.dataRate.get(inputStream =␣
↪repcap.InputStream.Default)


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
meas_data_rate: float Range: 0 to 999999999

6.18.3.10.1.3 FormatPy

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]:FORMat
```

class FormatPyCls

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

get(*inputStream=InputStream.Default*) → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]:FORMat
value: enums.CodingInputFormat = driver.source.bb.dvbs2.inputPy.isPy.formatPy.
↪get(inputStream = repcap.InputStream.Default)
```

Sets the format of the input signal.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
input_format: SMPTE| ASI

set(*input_format: CodingInputFormat*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]:FORMat
driver.source.bb.dvbs2.inputPy.isPy.formatPy.set(input_format = enums.
↪CodingInputFormat.ASI, inputStream = repcap.InputStream.Default)
```

Sets the format of the input signal.

param input_format
SMPTE| ASI

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.10.1.4 TsChannel

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]:TSCHannel
```

class TsChannelCls

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

get(*inputStream=InputStream.Default*) → NumberA

```
# SCPI: [SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]:TSCHannel
value: enums.NumberA = driver.source.bb.dvbs2.inputPy.isPy.tsChannel.
↪get(inputStream = repcap.InputStream.Default)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
ts_channel: 1| 2| 3| 4

set(*ts_channel: NumberA*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:INPut:[IS<CH>]:TSCHannel
driver.source.bb.dvbs2.inputPy.isPy.tsChannel.set(ts_channel = enums.NumberA._1,
↪ inputStream = repcap.InputStream.Default)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param ts_channel
1| 2| 3| 4

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.10.2 IsPy<InputStream>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.bb.dvbs2.isPy.repcap_inputStream_get()
driver.source.bb.dvbs2.isPy.repcap_inputStream_set(repcap.InputStream.Nr1)
```

class IsPyCls

IsPy commands group definition. 5 total commands, 4 Subgroups, 0 group commands Repeated Capability: InputStream, default value after init: InputStream.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.isPy.clone()
```

Subgroups

6.18.3.10.2.1 PacketLength

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[IS<CH>]:PACKetlength
```

class PacketLengthCls

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

get(*inputStream=InputStream.Default*) → InputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[IS<CH>]:PACKetlength
value: enums.InputSignalPacketLength = driver.source.bb.dvbs2.isPy.packetLength.
↪get(inputStream = repcap.InputStream.Default)
```

Queries the packet length of the external transport stream in bytes.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
packet_length: P188| P204| INValid P188|P204 188/204 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

6.18.3.10.2.2 Stuffing

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[IS<CH>]:STUFfing
```

class StuffingCls

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

get(*inputStream=InputStream.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[IS<CH>]:STUFfing
value: bool = driver.source.bb.dvbs2.isPy.stuffing.get(inputStream = repcap.
↪InputStream.Default)
```

Queries the stuffing state that is active.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
stuffing: 1| ON| 0| OFF

6.18.3.10.2.3 TestSignal

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[IS<CH>]:TESTsignal
```

class TestSignalCls

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

get(*inputStream=InputStream.Default*) → Dvbs2InputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[IS<CH>]:TESTsignal
value: enums.Dvbs2InputSignalTestSignal = driver.source.bb.dvbs2.isPy.
↪testSignal.get(inputStream = repcap.InputStream.Default)
```

Defines the test signal data.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
test_signal: TTSP| TGSP TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. TGSP Test GS packet with predefined packet data used as modulation data in the generic stream.

set(*test_signal: Dvbs2InputSignalTestSignal*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[IS<CH>]:TESTsignal
driver.source.bb.dvbs2.isPy.testSignal.set(test_signal = enums.
↪Dvbs2InputSignalTestSignal.TGSP, inputStream = repcap.InputStream.Default)
```

Defines the test signal data.

param test_signal
TTSP| TGSP TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. TGSP Test GS packet with predefined packet data used as modulation data in the generic stream.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.10.2.4 Useful

class UsefulCls

Useful 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.source.bb.dvbs2.isPy.useful.clone()
```

Subgroups

6.18.3.10.2.5 Rate

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[IS<CH>]:USEFul:[RaTE]
```

class RateCls

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

get(*inputStream=InputStream.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[IS<CH>]:USEFul:[RaTE]
value: float = driver.source.bb.dvbs2.isPy.useful.rate.get(inputStream = repcap.
↪InputStream.Default)
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
useful_data_rate: float Range: 0 to 999999999

set(*useful_data_rate: float*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[IS<CH>]:USEFul:[RaTE]
driver.source.bb.dvbs2.isPy.useful.rate.set(useful_data_rate = 1.0, inputStream␣
↪= repcap.InputStream.Default)
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

param useful_data_rate
float Range: 0 to 999999999

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.isPy.useful.rate.clone()
```

Subgroups

6.18.3.10.2.6 Max

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[IS<CH>]:USEFul:[RATE]:MAX
```

class MaxCls

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

get(*inputStream=InputStream.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[IS<CH>]:USEFul:[RATE]:MAX
value: float = driver.source.bb.dvbs2.isPy.useful.rate.max.get(inputStream =␣
↪repcap.InputStream.Default)
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
max_use_data_rate: float Range: 0 to 999999999

6.18.3.10.3 Prbs

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:PRBS:[SEQuence]
```

class PrbsCls

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

get_sequence() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PRBS:[SEQuence]
value: enums.SettingsPrbs = driver.source.bb.dvbs2.prbs.get_sequence()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
prbs: P15_1| P23_1

set_sequence(*prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:PRBS:[SEQuence]
driver.source.bb.dvbs2.prbs.set_sequence(prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param prbs
P15_1| P23_1

6.18.3.10.4 S2X

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS2:S2X:CHB
[SOURce<HW>]:BB:DVBS2:S2X:SF
[SOURce<HW>]:BB:DVBS2:S2X
```

class S2XCls

S2X commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_chb() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:S2X:CHB
value: bool = driver.source.bb.dvbs2.s2X.get_chb()
```

Enables or disables the chanel bonding.

return
chan_bonding: 1| ON| 0| OFF

get_sf() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:S2X:SF
value: bool = driver.source.bb.dvbs2.s2X.get_sf()
```

Enables or disables the super frame.

return
super_frame: 1| ON| 0| OFF

get_value() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:S2X
value: bool = driver.source.bb.dvbs2.s2X.get_value()
```

Enables S2-X features.

return
s_2_xstate: 1| ON| 0| OFF

set_chb(*chan_bonding: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:S2X:CHB
driver.source.bb.dvbs2.s2X.set_chb(chan_bonding = False)
```

Enables or disables the chanel bonding.

param chan_bonding
1| ON| 0| OFF

set_sf(*super_frame: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:S2X:SF
driver.source.bb.dvbs2.s2X.set_sf(super_frame = False)
```

Enables or disables the super frame.

param super_frame
1| ON| 0| OFF

set_value(*s_2_xstate: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:S2X
driver.source.bb.dvbs2.s2X.set_value(s_2_xstate = False)
```

Enables S2-X features.

param s_2_xstate
1| ON| 0| OFF

6.18.3.10.5 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS2:SETTing:CATalog
[SOURce<HW>]:BB:DVBS2:SETTing:DELete
[SOURce<HW>]:BB:DVBS2:SETTing:LOAD
[SOURce<HW>]:BB:DVBS2:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SETTing:DELete
driver.source.bb.dvbs2.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SETTing:CATalog
value: List[str] = driver.source.bb.dvbs2.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
dvbs_2_cat_name: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SETTing:LOAD
value: str = driver.source.bb.dvbs2.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: No help available

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SETTing:STORe
value: str = driver.source.bb.dvbs2.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.d2vbs) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: 'filename' Filename or complete file path

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SETTing:LOAD
driver.source.bb.dvbs2.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.d2vbs. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
'filename' Filename or complete file path; file extension can be omitted.

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SETTing:STORe
driver.source.bb.dvbs2.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.d2vbs) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
'filename' Filename or complete file path

6.18.3.10.6 Source

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:SOURce
```

class SourceCls

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

get_value() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.dvbs2.source.get_value()
```

Sets the modulation source for the input signal.

return
source: EXTernal| TSPLayer| TESTsignal

set_value(*source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SOURce
driver.source.bb.dvbs2.source.set_value(source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param source
EXTernal| TSPLayer| TESTsignal

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.source.clone()
```

Subgroups

6.18.3.10.6.1 IsPy

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:SOURce:IS<CH>
```

class IsPyCls

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

get(*inputStream=InputStream.Nr1*) → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SOURce:IS<CH>
value: enums.CodingInputSignalSource = driver.source.bb.dvbs2.source.isPy.
↪get(inputStream = repcap.InputStream.Nr1)
```

For VCM mode, queries the source for input streams 2 to 8. This source is always a test signal.

param inputStream
optional repeated capability selector. Default value: Nr1

return
source: TESTsignal

6.18.3.10.7 Special

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[SPECial]:GOLDcode
```

class SpecialCls

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

get_gold_code() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:GOLDcode
value: float = driver.source.bb.dvbs2.special.get_gold_code()
```

Defines the scrambling code number (n) of the gold code used for physical layer (PL) scrambling. This number in turn defines the scrambling sequence within a PL frame.

return
cold_code: No help available

set_gold_code(*cold_code: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:GOLDcode
driver.source.bb.dvbs2.special.set_gold_code(cold_code = 1.0)
```

Defines the scrambling code number (n) of the gold code used for physical layer (PL) scrambling. This number in turn defines the scrambling sequence within a PL frame.

param cold_code
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.special.clone()
```

Subgroups

6.18.3.10.7.1 DslPrbs

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[SPECial]:DSLPrbs:[STATe]
```

class DslPrbsCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:DSLPrbs:[STATe]
value: bool = driver.source.bb.dvbs2.special.dslPrbs.get_state()
```

Enable for test purposes. PRBS can be inserted into the data slices. The PRBS transmitted in the data slices is continuous, so that a BER measurement on decoded data slices can be performed.

return
data_slice_prbs: 1| ON| 0| OFF

set_state(*data_slice_prbs: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:DSLPrbs:[STATe]
driver.source.bb.dvbs2.special.dslPrbs.set_state(data_slice_prbs = False)
```

Enable for test purposes. PRBS can be inserted into the data slices. The PRBS transmitted in the data slices is continuous, so that a BER measurement on decoded data slices can be performed.

param data_slice_prbs
1| ON| 0| OFF

6.18.3.10.7.2 Scramble

SCPI Commands :

```
[SOURce<HW>]:BB:DVBS2:[SPECial]:SCRamble:SEQuence
[SOURce<HW>]:BB:DVBS2:[SPECial]:SCRamble:STATe
```

class ScrambleCls

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

get_sequence() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:SCRamble:SEQuence
value: float = driver.source.bb.dvbs2.special.scramble.get_sequence()
```

For normal applications, set this parameter to 0. If != 0 is set, the corresponding line of a hidden file is evaluated. PL header scrambling is performed, and the 'PL Gold Code Index (n) ' is set using the dedicated values of this line. The results are not displayed and are not readable. Also set the PL scrambling sequence ID in the DVB-S2 receiver.

return
scr_sequ: float Range: 0 to 9999

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:SCRamble:STATe
value: bool = driver.source.bb.dvbs2.special.scramble.get_state()
```

For test purposes, you can disable the PL scrambler. In normal operation it is enabled.

return
scrambler: 1| ON| 0| OFF

set_sequence(*scr_sequ: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:SCRamble:SEQuence
driver.source.bb.dvbs2.special.scramble.set_sequence(scr_sequ = 1.0)
```

For normal applications, set this parameter to 0. If != 0 is set, the corresponding line of a hidden file is evaluated. PL header scrambling is performed, and the 'PL Gold Code Index (n) ' is set using the dedicated values of this line. The results are not displayed and are not readable. Also set the PL scrambling sequence ID in the DVB-S2 receiver.

param scr_sequ
float Range: 0 to 9999

set_state(*scrambler: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:SCRamble:STATe
driver.source.bb.dvbs2.special.scramble.set_state(scrambler = False)
```

For test purposes, you can disable the PL scrambler. In normal operation it is enabled.

param scrambler
1| ON| 0| OFF

6.18.3.10.7.3 Setting

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:[SPECial]:SETTing:[STATe]
```

class SettingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:SETTing:[STATe]
value: bool = driver.source.bb.dvbs2.special.setting.get_state()
```

Enables or disables all special settings.

return
special_settings: 1| ON| 0| OFF

set_state(*special_settings: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:[SPECial]:SETTing:[STATe]
driver.source.bb.dvbs2.special.setting.set_state(special_settings = False)
```

Enables or disables all special settings.

param special_settings
1| ON| 0| OFF

6.18.3.10.8 Symbols

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:SYMBols:[RATE]
```

class SymbolsCls

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

get_rate() → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SYMBols:[RATE]
value: float = driver.source.bb.dvbs2.symbols.get_rate()
```

Sets the symbol rate. In the transmission spectrum, the symbol rate represents the 3 dB bandwidth.

return
symbol_rate: float Range: 0,100000 to 90,000000

set_rate(*symbol_rate: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:SYMBols:[RATE]
driver.source.bb.dvbs2.symbols.set_rate(symbol_rate = 1.0)
```

Sets the symbol rate. In the transmission spectrum, the symbol rate represents the 3 dB bandwidth.

param symbol_rate
float Range: 0,100000 to 90,000000

6.18.3.10.9 Tsl<TimeSlice>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.bb.dvbs2.tsl.repcap_timeSlice_get()
driver.source.bb.dvbs2.tsl.repcap_timeSlice_set(repcap.TimeSlice.Nr1)
```

class TslCls

Tsl commands group definition. 6 total commands, 1 Subgroups, 0 group commands Repeated Capability: TimeSlice, default value after init: TimeSlice.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.tsl.clone()
```

Subgroups

6.18.3.10.9.1 IsPy<InputStream>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.bb.dvbs2.tsl.isPy.repcap_inputStream_get()
driver.source.bb.dvbs2.tsl.isPy.repcap_inputStream_set(repcap.InputStream.Nr1)
```

class IsPyCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbs2.tsl.isPy.clone()
```

Subgroups

6.18.3.10.9.2 FecFrame

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:FECFrame
```

class FecFrameCls

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

get(*timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → Dvbs2CodingFecFrame

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:FECFrame
value: enums.Dvbs2CodingFecFrame = driver.source.bb.dvbs2.tsl.isPy.fecFrame.
↪get(timeSlice = repcap.TimeSlice.Default, inputStream = repcap.InputStream.
↪Default)
```

Sets the length of the FEC frames.

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
fec_frame: SHORt| NORMal| MEDium

set(*fec_frame: Dvbs2CodingFecFrame*, *timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:FECFrame
driver.source.bb.dvbs2.tsl.isPy.fecFrame.set(fec_frame = enums.
↪Dvbs2CodingFecFrame.MEDium, timeSlice = repcap.TimeSlice.Default, inputStream␣
↪= repcap.InputStream.Default)
```

Sets the length of the FEC frames.

param fec_frame
SHORt| NORMal| MEDium

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.10.9.3 Isi

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:ISI
```

class IsiCls

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

get(*timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:ISI
value: int = driver.source.bb.dvbs2.tsl.isPy.isi.get(timeSlice = repcap.
↪TimeSlice.Default, inputStream = repcap.InputStream.Default)
```

Sets the input stream identifier (ISI) .

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
isi: integer Range: 1 to 8

set(*isi: int*, *timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:ISI
driver.source.bb.dvbs2.tsl.isPy.isi.set(isi = 1, timeSlice = repcap.TimeSlice.
↪Default, inputStream = repcap.InputStream.Default)
```

Sets the input stream identifier (ISI) .

param isi
integer Range: 1 to 8

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.10.9.4 ModCod

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:MODCod
```

class ModCodCls

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

get(*timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → Dvbs2CodingModCod

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:MODCod
value: enums.Dvbs2CodingModCod = driver.source.bb.dvbs2.tsl.isPy.modCod.
↪get(timeSlice = repcap.TimeSlice.Default, inputStream = repcap.InputStream.
↪Default)
```

Defines the modulation coding, a combined setting of constellation and code rate.

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
mod_cod: 0| 1| 10| 100| 101| 102| 103| 104| 105| 106| 11| 12| 13| 14| 15| 16| 17| 18| 19| 2| 20| 21| 22| 23| 24| 25| 26| 27| 28| 29| 3| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 4| 40| 41| 42| 43| 44| 45| 46| 47| 48| 49| 5| 50| 51| 52| 53| 54| 55| 56| 57| 58| 59| 6| 60| 61| 62| 63| 64| 65| 66| 67| 68| 69| 7| 70| 71| 72| 73| 74| 75| 76| 77| 78| 79| 8| 80| 81| 82| 83| 84| 85| 86| 87| 88| 89| 9| 90| 91| 92| 93| 94| 95| 96| 97| 98| 99

set(*mod_cod: Dvbs2CodingModCod*, *timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:MODCod
driver.source.bb.dvbs2.tsl.isPy.modCod.set(mod_cod = enums.Dvbs2CodingModCod._0,
↪ timeSlice = repcap.TimeSlice.Default, inputStream = repcap.InputStream.
↪Default)
```

Defines the modulation coding, a combined setting of constellation and code rate.

param mod_cod
0| 1| 10| 100| 101| 102| 103| 104| 105| 106| 11| 12| 13| 14| 15| 16| 17| 18| 19| 2| 20| 21| 22| 23| 24| 25| 26| 27| 28| 29| 3| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 4| 40| 41| 42| 43| 44| 45| 46| 47| 48| 49| 5| 50| 51| 52| 53| 54| 55| 56| 57| 58| 59| 6| 60| 61| 62| 63| 64| 65|

66| 67| 68| 69| 7| 70| 71| 72| 73| 74| 75| 76| 77| 78| 79| 8| 80| 81| 82| 83| 84| 85| 86| 87| 88| 89| 9| 90| 91| 92| 93| 94| 95| 96| 97| 98| 99

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.10.9.5 Nsym

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:NSYM
```

class NsymCls

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

get(*timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:NSYM
value: int = driver.source.bb.dvbs2.tsl.isPy.nsym.get(timeSlice = repcap.
↪TimeSlice.Default, inputStream = repcap.InputStream.Default)
```

Displays the information about the number of symbols.

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
num_sym: integer Range: 0 to 99999

6.18.3.10.9.6 Pilots

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:PILots
```

class PilotsCls

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

get(*timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:PILots
value: bool = driver.source.bb.dvbs2.tsl.isPy.pilots.get(timeSlice = repcap.
↪TimeSlice.Default, inputStream = repcap.InputStream.Default)
```

Controls the insertion of pilot symbols during the formation of the physical layer frame. Pilot symbols generate an unmodulated carrier and are helpful for synchronizing receivers under difficult transmission conditions.

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
pilots: 1| ON| 0| OFF

set(*pilots: bool*, *timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:PILots
driver.source.bb.dvbs2.tsl.isPy.pilots.set(pilots = False, timeSlice = repcap.
↪TimeSlice.Default, inputStream = repcap.InputStream.Default)
```

Controls the insertion of pilot symbols during the formation of the physical layer frame. Pilot symbols generate an unmodulated carrier and are helpful for synchronizing receivers under difficult transmission conditions.

param pilots
1| ON| 0| OFF

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.10.9.7 Tsn

SCPI Command :

```
[SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:TSN
```

class TsnCls

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

get(*timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:TSN
value: float = driver.source.bb.dvbs2.tsl.isPy.tsn.get(timeSlice = repcap.
↪TimeSlice.Default, inputStream = repcap.InputStream.Default)
```

Sets the time slice number (TSN) or the input stream identifier (ISI) in hexadecimal representation. This number is used for identification. Each time slice uses a unique TSN.

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

return
tsn: float

set(*tsn: float*, *timeSlice=TimeSlice.Default*, *inputStream=InputStream.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBS2:TSL<ST>:IS<CH>:TSN
driver.source.bb.dvbs2.tsl.isPy.tsn.set(tsn = 1.0, timeSlice = repcap.TimeSlice.
↪Default, inputStream = repcap.InputStream.Default)
```

Sets the time slice number (TSN) or the input stream identifier (ISI) in hexadecimal representation. This number is used for identification. Each time slice uses a unique TSN.

param tsn
float

param timeSlice
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tsl')

param inputStream
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IsPy')

6.18.3.11 Dvbt

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:CONStel
[SOURce<HW>]:BB:DVBT:DVHState
[SOURce<HW>]:BB:DVBT:HIERarchy
[SOURce<HW>]:BB:DVBT:PAYLoad
[SOURce<HW>]:BB:DVBT:PID
[SOURce<HW>]:BB:DVBT:PIDTestpack
[SOURce<HW>]:BB:DVBT:PRESet
[SOURce<HW>]:BB:DVBT:STATe
[SOURce<HW>]:BB:DVBT:TSPacket
```

class DvbtCls

Dvbt commands group definition. 51 total commands, 19 Subgroups, 9 group commands

get_constel() → DvbtCodingConstel

```
# SCPI: [SOURce<HW>]:BB:DVBT:CONStel
value: enums.DvbtCodingConstel = driver.source.bb.dvbt.get_constel()
```

Defines the constellation.

return
constel: T64| T16| T4

get_dvh_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:DVHState
value: bool = driver.source.bb.dvbt.get_dvh_state()
```

Enables or disables .

return
dvh_state: 1| ON| 0| OFF

get_hierarchy() → DvbtCodingHierarchy

```
# SCPI: [SOURce<HW>]:BB:DVBT:HIERarchy
value: enums.DvbtCodingHierarchy = driver.source.bb.dvbt.get_hierarchy()
```

Selects the coding hierarchy.

return
hierarchy: A4| A2| A1| NONHier

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:DVBT:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.dvbt.get_payload()
```

Defines the payload area content of the packet.

return
payload: HFF| H00| PRBS

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:DVBT:PID
value: int = driver.source.bb.dvbt.get_pid()
```

Sets the .

return
pid: integer Range: #H0000 to #H1FFF

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:DVBT:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.dvbt.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
pid_test_packet: NULL| VARiable

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:STATe
value: bool = driver.source.bb.dvbt.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:DVBT:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.dvbt.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
ts_packet: S187| H184

preset() → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:PRESet
driver.source.bb.dvbt.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>:BB:DVBT:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:DVBT:PRESet
driver.source.bb.dvbt.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>:BB:DVBT:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_constel(*constel: DvbtCodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:CONStel
driver.source.bb.dvbt.set_constel(constel = enums.DvbtCodingConstel.T16)
```

Defines the constellation.

param constel
T64| T16| T4

set_dvh_state(*dvh_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:DVHState
driver.source.bb.dvbt.set_dvh_state(dvh_state = False)
```

Enables or disables .

param dvh_state
1| ON| 0| OFF

set_hierarchy(*hierarchy: DvbtCodingHierarchy*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:HIERarchy
driver.source.bb.dvbt.set_hierarchy(hierarchy = enums.DvbtCodingHierarchy.A1)
```

Selects the coding hierarchy.

param hierarchy
A4| A2| A1| NONHier

set_payload(*payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:PAYLoad
driver.source.bb.dvbt.set_payload(payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param payload
HFF| H00| PRBS

set_pid(*pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:PID
driver.source.bb.dvbt.set_pid(pid = 1)
```

Sets the .

param pid
integer Range: #H0000 to #H1FFF

set_pid_test_pack(*pid_test_packet: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:PIDTestpack
driver.source.bb.dvbt.set_pid_test_pack(pid_test_packet = enums.PidTestPacket.
↪NULL)
```

If a header is present in the test packet ('Test TS Packet > Head/184 Payload') , you can specify a fixed or variable packet identifier (PID) .

param pid_test_packet
NULL| VARiable

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:STATe
driver.source.bb.dvbt.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

set_ts_packet(*ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:TSPacket
driver.source.bb.dvbt.set_ts_packet(ts_packet = enums.SettingsTestTsPacket.H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param ts_packet
S187| H184

Cloning the Group

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

Subgroups

6.18.3.11.1 Cell

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:CELL:ID
```

class CellCls

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

get_id() → int

```
# SCPI: [SOURce<HW>]:BB:DVBT:CELL:ID
value: int = driver.source.bb.dvbt.cell.get_id()
```

Sets the cell ID in four-digit hexadecimal format.

return
cell_id: integer Range: #H0000 to #HFFFF

set_id(*cell_id: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:CELL:ID
driver.source.bb.dvbt.cell.set_id(cell_id = 1)
```

Sets the cell ID in four-digit hexadecimal format.

param cell_id
integer Range: #H0000 to #HFFFF

6.18.3.11.2 Channel

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:CHANnel:[BANDwidth]
```

class ChannelCls

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

get_bandwidth() → DvbtCodingChannelBandwidth

```
# SCPI: [SOURce<HW>]:BB:DVBT:CHANnel:[BANDwidth]
value: enums.DvbtCodingChannelBandwidth = driver.source.bb.dvbt.channel.get_
↪bandwidth()
```

Selects the channel bandwidth.

return
channel_bw: BW_Var| BW_8| BW_7| BW_5| BW_6

set_bandwidth(*channel_bw: DvbtCodingChannelBandwidth*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:CHANnel:[BANDwidth]
driver.source.bb.dvbt.channel.set_bandwidth(channel_bw = enums.
↪DvbtCodingChannelBandwidth.BW_5)
```

Selects the channel bandwidth.

param channel_bw
BW_Var| BW_8| BW_7| BW_5| BW_6

6.18.3.11.3 Dvbh

class DvbhCls

Dvbh 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.bb.dvbt.dvbh.clone()
```

Subgroups

6.18.3.11.3.1 Symbol

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:DVBH:SYMBol:[INTerleaver]
```

class SymbolCls

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

get_interleaver() → DvbtCodingDvbhSymbolInterleaver

```
# SCPI: [SOURce<HW>]:BB:DVBT:DVBH:SYMBol:[INTerleaver]
value: enums.DvbtCodingDvbhSymbolInterleaver = driver.source.bb.dvbt.dvbh.
↪symbol.get_interleaver()
```

Sets the symbol interleaver.

return
symb_interleaver: INDepth| NATive

set_interleaver(*symb_interleaver: DvbtCodingDvbhSymbolInterleaver*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:DVBH:SYMBol:[INTerleaver]
driver.source.bb.dvbt.dvbh.symbol.set_interleaver(symb_interleaver = enums.
↪DvbtCodingDvbhSymbolInterleaver.INDepth)
```

Sets the symbol interleaver.

param symb_interleaver
INDepth| NATive

6.18.3.11.4 Fft

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:FFT:MODE
```

class FftCls

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

get_mode() → DvbtCodingFftMode

```
# SCPI: [SOURce<HW>]:BB:DVBT:FFT:MODE
value: enums.DvbtCodingFftMode = driver.source.bb.dvbt.fft.get_mode()
```

Sets the fast fourier transform (FFT) window to determine the number of carriers per OFDM symbol. To find out the number of carriers for a given FFT window, see Table 'Number of carriers'.

return
fft_mode: M8K| M4K| M2K

set_mode(*fft_mode: DvbtCodingFftMode*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:FFT:MODE
driver.source.bb.dvbt.fft.set_mode(fft_mode = enums.DvbtCodingFftMode.M2K)
```

Sets the fast fourier transform (FFT) window to determine the number of carriers per OFDM symbol. To find out the number of carriers for a given FFT window, see Table 'Number of carriers'.

param fft_mode
M8K| M4K| M2K

6.18.3.11.5 Guard

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:GUARd:INTerval
```

class GuardCls

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

get_interval() → DvbtCodingGuardInterval

```
# SCPI: [SOURce<HW>]:BB:DVBT:GUARd:INTerval
value: enums.DvbtCodingGuardInterval = driver.source.bb.dvbt.guard.get_
↪interval()
```

Sets the guard interval. The interval is expressed in fractions of the useful part of the OFDM symbol period Tu.

return
guard_int: G1_32| G1_16| G1| G1_| G1_8| G1_4

set_interval(*guard_int: DvbtCodingGuardInterval*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:GUARd:INTerval
driver.source.bb.dvbt.guard.set_interval(guard_int = enums.
↪DvbtCodingGuardInterval.G1)
```

Sets the guard interval. The interval is expressed in fractions of the useful part of the OFDM symbol period Tu.

param guard_int
G1_32| G1_16| G1| G1_| G1_8| G1_4

6.18.3.11.6 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:INPut:LOW
[SOURce<HW>]:BB:DVBT:INPut:[HIGH]
```

class InputPyCls

InputPy commands group definition. 8 total commands, 3 Subgroups, 2 group commands

get_high() → CodingInputSignalInputB

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:[HIGH]
value: enums.CodingInputSignalInputB = driver.source.bb.dvbt.inputPy.get_high()
```

Sets the external input interface.

return
input_py: IP| TS TS Input for serial transport stream data. The signal is input at the 'User 1/2' connectors. IP Supported for high priority path (HP) only, i.e. setting requires non-hierarchical coding. Input for IP transport stream data. The signal is input at the 'IP Data' connector.

get_low() → CodingInputSignalInputB

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:LOW
value: enums.CodingInputSignalInputB = driver.source.bb.dvbt.inputPy.get_low()
```

Sets the external input interface.

return
input_lp: No help available

set_high(*input_py: CodingInputSignalInputB*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:[HIGH]
driver.source.bb.dvbt.inputPy.set_high(input_py = enums.CodingInputSignalInputB.
↪ASIFront)
```

Sets the external input interface.

param input_py
IP| TS TS Input for serial transport stream data. The signal is input at the 'User 1/2'

connectors. IP Supported for high priority path (HP) only, i.e. setting requires non-hierarchical coding. Input for IP transport stream data. The signal is input at the 'IP Data' connector.

set_low(*input_lp: CodingInputSignalInputB*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:LOW
driver.source.bb.dvbt.inputPy.set_low(input_lp = enums.CodingInputSignalInputB.
↪ASIFront)
```

Sets the external input interface.

param input_lp
IP| TS TS Input for serial transport stream data. The signal is input at the 'User 1/2' connectors. IP Supported for high priority path (HP) only, i.e. setting requires non-hierarchical coding. Input for IP transport stream data. The signal is input at the 'IP Data' connector.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbt.inputPy.clone()
```

Subgroups

6.18.3.11.6.1 DataRate

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:[INPut]:DATarate:LOW
[SOURce<HW>]:BB:DVBT:[INPut]:DATarate:[HIGH]
```

class DataRateCls

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

get_high() → float

```
# SCPI: [SOURce<HW>]:BB:DVBT:[INPut]:DATarate:[HIGH]
value: float = driver.source.bb.dvbt.inputPy.dataRate.get_high()


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
meas_drhp: float Range: 0 to 9999999999

get_low() → float

```
# SCPI: [SOURce<HW>]:BB:DVBT:[INPut]:DATarate:LOW
value: float = driver.source.bb.dvbt.inputPy.dataRate.get_low()


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
meas_drlp: float Range: 0 to 9999999999

6.18.3.11.6.2 FormatPy

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:INPut:FORMat:LOW
[SOURce<HW>]:BB:DVBT:INPut:FORMat
```

class FormatPyCls

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

get_low() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:FORMat:LOW
value: enums.CodingInputFormat = driver.source.bb.dvbt.inputPy.formatPy.get_
↪low()
```

Sets the format of the input signal.

return
input_format_lp: No help available

get_value() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.dvbt.inputPy.formatPy.get_
↪value()
```

Sets the format of the input signal.

return
input_format: ASI| SMPTE

set_low(*input_format_lp: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:FORMat:LOW
driver.source.bb.dvbt.inputPy.formatPy.set_low(input_format_lp = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param input_format_lp
ASI| SMPTE

set_value(*input_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:FORMat
driver.source.bb.dvbt.inputPy.formatPy.set_value(input_format = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param input_format
ASI| SMPTE

6.18.3.11.6.3 TsChannel

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:INPut:TSCHannel:LOW
[SOURce<HW>]:BB:DVBT:INPut:TSCHannel:[HIGH]
```

class TsChannelCls

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

get_high() → NumberA

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:TSCHannel:[HIGH]
value: enums.NumberA = driver.source.bb.dvbt.inputPy.tsChannel.get_high()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

return
ts_channel: 1| 2| 3| 4

get_low() → NumberA

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:TSCHannel:LOW
value: enums.NumberA = driver.source.bb.dvbt.inputPy.tsChannel.get_low()
```

No command help available

return
ts_channel_lp: No help available

set_high(*ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:TSCHannel:[HIGH]
driver.source.bb.dvbt.inputPy.tsChannel.set_high(ts_channel = enums.NumberA._1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param ts_channel
1| 2| 3| 4

set_low(*ts_channel_lp: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:INPut:TSCHannel:LOW
driver.source.bb.dvbt.inputPy.tsChannel.set_low(ts_channel_lp = enums.NumberA._
↪1)
```

No command help available

param ts_channel_lp
No help available

6.18.3.11.7 MpeFec

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:MPEFec:LOW
[SOURce<HW>]:BB:DVBT:MPEFec:[HIGH]
```

class MpeFecCls

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

get_high() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:MPEFec:[HIGH]
value: bool = driver.source.bb.dvbt.mpeFec.get_high()
```

Enables/disables . If enabled, 1 TPS bit (s49) is used to signal that MPE FEC is used in at least one data stream.

return
mpe_fec_hp: 1| ON| 0| OFF

get_low() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:MPEFec:LOW
value: bool = driver.source.bb.dvbt.mpeFec.get_low()
```

Enables/disables . If enabled, 1 TPS bit (s49) is used to signal that MPE FEC is used in at least one data stream.

return
mpe_fec_lp: No help available

set_high(*mpe_fec_hp: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:MPEFec:[HIGH]
driver.source.bb.dvbt.mpeFec.set_high(mpe_fec_hp = False)
```

Enables/disables . If enabled, 1 TPS bit (s49) is used to signal that MPE FEC is used in at least one data stream.

param mpe_fec_hp
1| ON| 0| OFF

set_low(*mpe_fec_lp: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:MPEFec:LOW
driver.source.bb.dvbt.mpeFec.set_low(mpe_fec_lp = False)
```

Enables/disables . If enabled, 1 TPS bit (s49) is used to signal that MPE FEC is used in at least one data stream.

param mpe_fec_lp
1| ON| 0| OFF

6.18.3.11.8 PacketLength

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:PACKetlength:LOW
[SOURce<HW>]:BB:DVBT:PACKetlength:[HIGH]
```

class PacketLengthCls

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

get_high() → InputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:DVBT:PACKetlength:[HIGH]
value: enums.InputSignalPacketLength = driver.source.bb.dvbt.packetLength.get_
↪high()
```

Queries the packet length of the external transport stream in bytes.

return
packet_length_hp: P188| P204| INValid P188|P204 188/204 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

get_low() → InputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:DVBT:PACKetlength:LOW
value: enums.InputSignalPacketLength = driver.source.bb.dvbt.packetLength.get_
↪low()
```

Queries the packet length of the external transport stream in bytes.

return
packet_length_lp: P188| P204| INValid P188|P204 188/204 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

6.18.3.11.9 Prbs

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:PRBS:[SEQuence]
```

class PrbsCls

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

get_sequence() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:DVBT:PRBS:[SEQuence]
value: enums.SettingsPrbs = driver.source.bb.dvbt.prbs.get_sequence()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
prbs: P15_1| P23_1

set_sequence(*prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:PRBS:[SEQuence]
driver.source.bb.dvbt.prbs.set_sequence(prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param prbs
P15_1| P23_1

6.18.3.11.10 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:RATE:LOW
[SOURce<HW>]:BB:DVBT:RATE:[HIGH]
```

class RateCls

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

get_high() → CodingCoderate

```
# SCPI: [SOURce<HW>]:BB:DVBT:RATE:[HIGH]
value: enums.CodingCoderate = driver.source.bb.dvbt.rate.get_high()
```

Sets the code rate.

return
coderate_hp: R1_2| R2_3| R3_4| R5_6| R7_8

get_low() → CodingCoderate

```
# SCPI: [SOURce<HW>]:BB:DVBT:RATE:LOW
value: enums.CodingCoderate = driver.source.bb.dvbt.rate.get_low()
```

Sets the code rate.

return
coderate_lp: No help available

set_high(*coderate_hp: CodingCoderate*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:RATE:[HIGH]
driver.source.bb.dvbt.rate.set_high(coderate_hp = enums.CodingCoderate.R1_2)
```

Sets the code rate.

param coderate_hp
R1_2| R2_3| R3_4| R5_6| R7_8

set_low(*coderate_lp: CodingCoderate*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:RATE:LOW
driver.source.bb.dvbt.rate.set_low(coderate_lp = enums.CodingCoderate.R1_2)
```

Sets the code rate.

param coderate_lp
R1_2| R2_3| R3_4| R5_6| R7_8

6.18.3.11.11 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:SETTing:CATalog
[SOURce<HW>]:BB:DVBT:SETTing:DELete
[SOURce<HW>]:BB:DVBT:SETTing:LOAD
[SOURce<HW>]:BB:DVBT:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:SETTing:DELete
driver.source.bb.dvbt.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.dvbt. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:DVBT:SETTing:CATalog
value: List[str] = driver.source.bb.dvbt.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.dvbt. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
dvbt_cat_name: No help available

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:DVBT:SETTing:LOAD
value: str = driver.source.bb.dvbt.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dvbt. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
dvbt_recall: 'filename' Filename or complete file path; file extension can be omitted

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:DVBT:SETTing:STORe
value: str = driver.source.bb.dvbt.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.dvbt) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
dvbt_save: 'filename'

set_load(*dvbt_recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:SETTing:LOAD
driver.source.bb.dvbt.setting.set_load(dvbt_recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dvbt. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param dvbt_recall
'filename' Filename or complete file path; file extension can be omitted

set_store(*dvbt_save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:SETTing:STORe
driver.source.bb.dvbt.setting.set_store(dvbt_save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.dvbt) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param dvbt_save
'filename'

6.18.3.11.12 Source

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:SOURce:LOW
[SOURce<HW>]:BB:DVBT:SOURce:[HIGH]
```

class SourceCls

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

get_high() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DVBT:SOURce:[HIGH]
value: enums.CodingInputSignalSource = driver.source.bb.dvbt.source.get_high()
```

Sets the modulation source for the input signal.

return
signal_source_hp: TSPLayer| EXTernal| TESTsignal

get_low() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:DVBT:SOURce:LOW
value: enums.CodingInputSignalSource = driver.source.bb.dvbt.source.get_low()
```

Sets the modulation source for the input signal.

return
source_lp: No help available

set_high(*signal_source_hp: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:SOURce:[HIGH]
driver.source.bb.dvbt.source.set_high(signal_source_hp = enums.
↪CodingInputSignalSource.EXTernal)
```

Sets the modulation source for the input signal.

param signal_source_hp
TSPLayer| EXTernal| TESTsignal

set_low(*source_lp: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:SOURce:LOW
driver.source.bb.dvbt.source.set_low(source_lp = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param source_lp
TSPLayer| EXTernal| TESTsignal

6.18.3.11.13 Special

class SpecialCls

Special 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.bb.dvbt.special.clone()
```

Subgroups

6.18.3.11.13.1 ReedSolomon

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:[SPECial]:REEDsolomon:LOW
[SOURce<HW>]:BB:DVBT:[SPECial]:REEDsolomon:[HIGH]
```

class ReedSolomonCls

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

get_high() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:[SPECial]:REEDsolomon:[HIGH]
value: bool = driver.source.bb.dvbt.special.reedSolomon.get_high()
```

Enables/disables the Reed-Solomon encoder.

return
reed_solomon_high: OFF| ON| 1| 0

get_low() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:[SPECial]:REEDsolomon:LOW
value: bool = driver.source.bb.dvbt.special.reedSolomon.get_low()
```

Enables/disables the Reed-Solomon encoder.

return
reed_solomon_low: No help available

set_high(*reed_solomon_high: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:[SPECial]:REEDsolomon:[HIGH]
driver.source.bb.dvbt.special.reedSolomon.set_high(reed_solomon_high = False)
```

Enables/disables the Reed-Solomon encoder.

param reed_solomon_high
OFF| ON| 1| 0

set_low(*reed_solomon_low: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:[SPECial]:REEDsolomon:LOW
driver.source.bb.dvbt.special.reedSolomon.set_low(reed_solomon_low = False)
```

Enables/disables the Reed-Solomon encoder.

param reed_solomon_low
OFF| ON| 1| 0

6.18.3.11.13.2 Setting

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:[SPECial]:SETTing:[STATe]
```

class SettingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:[SPECial]:SETTing:[STATe]
value: bool = driver.source.bb.dvbt.special.setting.get_state()
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

return
settings: 1| ON| 0| OFF

set_state(*settings: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:[SPECial]:SETTing:[STATe]
driver.source.bb.dvbt.special.setting.set_state(settings = False)
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

param settings
1| ON| 0| OFF

6.18.3.11.14 Stuffing

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:STUFfing:LOW
[SOURce<HW>]:BB:DVBT:STUFfing:[HIGH]
```

class StuffingCls

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

get_high() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:STUFfing:[HIGH]
value: bool = driver.source.bb.dvbt.stuffing.get_high()
```

Queries the stuffing state that is active for HP path and LP path.

return
stuffing_hp: 1| ON| 0| OFF

get_low() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:STUFfing:LOW
value: bool = driver.source.bb.dvbt.stuffing.get_low()
```

Queries the stuffing state that is active for HP path and LP path.

return
stuffing_lp: 1| ON| 0| OFF

6.18.3.11.15 TestSignal

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:TESTsignal:LOW
[SOURce<HW>]:BB:DVBT:TESTsignal:[HIGH]
```

class TestSignalCls

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

get_high() → InputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:DVBT:TESTsignal:[HIGH]
value: enums.InputSignalTestSignal = driver.source.bb.dvbt.testSignal.get_high()
```

Defines the test signal data.

return
test_signal_hp: TTSP| PBEC| PAFC TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBEC PRBS before convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the convolutional encoder. PRBS data conforms with specification. PAFC PRBS after convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted after the convolutional encoder.

get_low() → InputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:DVBT:TESTsignal:LOW
value: enums.InputSignalTestSignal = driver.source.bb.dvbt.testSignal.get_low()
```

Defines the test signal data.

return
test_signal_lp: No help available

set_high(*test_signal_hp: InputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:TESTsignal:[HIGH]
driver.source.bb.dvbt.testSignal.set_high(test_signal_hp = enums.
↪InputSignalTestSignal.PAFC)
```

Defines the test signal data.

param test_signal_hp
TTSP| PBEC| PAFC TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBEC PRBS before convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the convolutional encoder. PRBS data conforms with specification. PAFC PRBS after convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted after the convolutional encoder.

set_low(*test_signal_lp: InputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:TESTsignal:LOW
driver.source.bb.dvbt.testSignal.set_low(test_signal_lp = enums.
↪InputSignalTestSignal.PAFC)
```

Defines the test signal data.

param test_signal_lp
TTSP| PBEC| PAFC TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBEC PRBS before convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the convolutional encoder. PRBS data conforms with specification. PAFC PRBS after convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted after the convolutional encoder.

6.18.3.11.16 Timeslice

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:TIMeslice:LOW
[SOURce<HW>]:BB:DVBT:TIMeslice:[HIGH]
```

class TimesliceCls

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

get_high() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:TIMeslice:[HIGH]
value: bool = driver.source.bb.dvbt.timeslice.get_high()
```

Enables/disables time slicing. If enabled, 1 TPS bit (s48) is used to signal that at least one data stream with time slicing exists in the multiplex.

return
time_slicing_hp: 1| ON| 0| OFF

get_low() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:TIMeslice:LOW
value: bool = driver.source.bb.dvbt.timeslice.get_low()
```

Enables/disables time slicing. If enabled, 1 TPS bit (s48) is used to signal that at least one data stream with time slicing exists in the multiplex.

return
time_slicing_lp: No help available

set_high(*time_slicing_hp: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:TIMeslice:[HIGH]
driver.source.bb.dvbt.timeslice.set_high(time_slicing_hp = False)
```

Enables/disables time slicing. If enabled, 1 TPS bit (s48) is used to signal that at least one data stream with time slicing exists in the multiplex.

param time_slicing_hp
1| ON| 0| OFF

set_low(*time_slicing_lp: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:TIMeslice:LOW
driver.source.bb.dvbt.timeslice.set_low(time_slicing_lp = False)
```

Enables/disables time slicing. If enabled, 1 TPS bit (s48) is used to signal that at least one data stream with time slicing exists in the multiplex.

param time_slicing_lp
1| ON| 0| OFF

6.18.3.11.17 TpsReserved

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:TPSReserved:STATE
[SOURce<HW>]:BB:DVBT:TPSReserved:VALue
```

class TpsReservedCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:DVBT:TPSReserved:STATE
value: bool = driver.source.bb.dvbt.tpsReserved.get_state()
```

Enables or disables the reserved TPS bits.

return
tps_reserved: 1| ON| 0| OFF

get_value() → int

```
# SCPI: [SOURce<HW>]:BB:DVBT:TPSReserved:VALue
value: int = driver.source.bb.dvbt.tpsReserved.get_value()
```

Sets the reserved bits in one-digit hexadecimal format.

return
reserved_bits: integer Range: #H0 to #HF

set_state(*tps_reserved: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:TPSReserved:STATE
driver.source.bb.dvbt.tpsReserved.set_state(tps_reserved = False)
```

Enables or disables the reserved TPS bits.

param tps_reserved
1| ON| 0| OFF

set_value(*reserved_bits: int*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:TPSReserved:VALue
driver.source.bb.dvbt.tpsReserved.set_value(reserved_bits = 1)
```

Sets the reserved bits in one-digit hexadecimal format.

param reserved_bits
integer Range: #H0 to #HF

6.18.3.11.18 Used

SCPI Command :

```
[SOURce<HW>]:BB:DVBT:USED:[BANDwidth]
```

class UsedCls

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

get_bandwidth() → float

```
# SCPI: [SOURce<HW>]:BB:DVBT:USED:[BANDwidth]
value: float = driver.source.bb.dvbt.used.get_bandwidth()
```

Defines the used bandwidth.

return
used_bw: float Range: 1000000 to 10000000

set_bandwidth(*used_bw: float*) → None

```
# SCPI: [SOURce<HW>]:BB:DVBT:USED:[BANDwidth]
driver.source.bb.dvbt.used.set_bandwidth(used_bw = 1.0)
```

Defines the used bandwidth.

param used_bw
float Range: 1000000 to 10000000

6.18.3.11.19 Useful

class UsefulCls

Useful 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.bb.dvbt.useful.clone()
```

Subgroups

6.18.3.11.19.1 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:USEFul:[RATE]:LOW
[SOURce<HW>]:BB:DVBT:USEFul:[RATE]:[HIGH]
```

class RateCls

Rate commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_high() → float

```
# SCPI: [SOURce<HW>]:BB:DVBT:USEFul:[RATE]:[HIGH]
value: float = driver.source.bb.dvbt.useful.rate.get_high()
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

return
use_drhp: float Range: 0 to 9999999999

get_low() → float

```
# SCPI: [SOURce<HW>]:BB:DVBT:USEFul:[RATE]:LOW
value: float = driver.source.bb.dvbt.useful.rate.get_low()
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

return
use_drlp: float Range: 0 to 9999999999

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.dvbt.useful.rate.clone()
```

Subgroups

6.18.3.11.19.2 Max

SCPI Commands :

```
[SOURce<HW>]:BB:DVBT:USEFul:[RATE]:MAX:LOW
[SOURce<HW>]:BB:DVBT:USEFul:[RATE]:MAX:[HIGH]
```

class MaxCls

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

get_high() → int

```
# SCPI: [SOURce<HW>]:BB:DVBT:USEFul:[RATE]:MAX:[HIGH]
value: int = driver.source.bb.dvbt.useful.rate.max.get_high()
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

return
max_use_drhp: integer Range: 0 to 9999999999

get_low() → int

```
# SCPI: [SOURce<HW>]:BB:DVBT:USEFul:[RATE]:MAX:LOW
value: int = driver.source.bb.dvbt.useful.rate.max.get_low()
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

return
max_use_drlp: integer Range: 0 to 9999999999

6.18.3.12 General

class GeneralCls

General commands group definition. 23 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.18.3.12.1 Am

SCPI Commands :

```
[SOURce<HW>]:BB:GENeral:AM:DEPTh
[SOURce<HW>]:BB:GENeral:AM:FREQuency
[SOURce<HW>]:BB:GENeral:AM:PERiod
[SOURce<HW>]:BB:GENeral:AM:SHAPe
[SOURce<HW>]:BB:GENeral:AM:[STATe]
```

class AmCls

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

get_depth() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:DEPTh
value: float = driver.source.bb.general.am.get_depth()
```

Sets the depth of the modulation signal in percent. The depth is limited by the maximum peak envelope power (PEP) .

return
am_depth: float Range: 0 to 100

get_frequency() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:FREQuency
value: float = driver.source.bb.general.am.get_frequency()
```

Sets the frequency of the modulation signal.

return
am_freq: float Range: 0.1 to 100E3

get_period() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:PERiod
value: float = driver.source.bb.general.am.get_period()
```

Queries the period of the modulation signal.

return
am_per: float Range: 100E-9 to 100

get_shape() → BasebandModShape

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:SHAPe
value: enums.BasebandModShape = driver.source.bb.general.am.get_shape()
```

Queries the shape of the modulation signal.

return
am_shape: SINE

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:[STATe]
value: bool = driver.source.bb.general.am.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
am_mod_state: 1| ON| 0| OFF

set_depth(*am_depth: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:DEPTh
driver.source.bb.general.am.set_depth(am_depth = 1.0)
```

Sets the depth of the modulation signal in percent. The depth is limited by the maximum peak envelope power (PEP) .

param am_depth
float Range: 0 to 100

set_frequency(*am_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:FREQuency
driver.source.bb.general.am.set_frequency(am_freq = 1.0)
```

Sets the frequency of the modulation signal.

param am_freq
float Range: 0.1 to 100E3

set_period(*am_per: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:PERiod
driver.source.bb.general.am.set_period(am_per = 1.0)
```

Queries the period of the modulation signal.

param am_per
float Range: 100E-9 to 100

set_shape(*am_shape: BasebandModShape*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:SHAPe
driver.source.bb.general.am.set_shape(am_shape = enums.BasebandModShape.SINE)
```

Queries the shape of the modulation signal.

param am_shape
SINE

set_state(*am_mod_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:AM:[STATe]
driver.source.bb.general.am.set_state(am_mod_state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param am_mod_state
1| ON| 0| OFF

6.18.3.12.2 Fm

SCPI Commands :

```
[SOURce<HW>]:BB:GENeral:FM:DEViation
[SOURce<HW>]:BB:GENeral:FM:FREQuency
[SOURce<HW>]:BB:GENeral:FM:PERiod
[SOURce<HW>]:BB:GENeral:FM:SHAPe
[SOURce<HW>]:BB:GENeral:FM:[STATe]
```

class FmCls

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

get_deviation() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:DEViation
value: float = driver.source.bb.general.fm.get_deviation()
```

Sets the frequency modulation deviation in Hz.

return
fm_deviation: float Range: 0 to 4E6

get_frequency() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:FREQuency
value: float = driver.source.bb.general.fm.get_frequency()
```

Sets the frequency of the modulation signal.

return
freq_mod_freq: float Range: 0.1 to 100E3

get_period() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:PERiod
value: float = driver.source.bb.general.fm.get_period()
```

Queries the period of the modulation signal.

return
fm_per: float Range: 100E-9 to 100

get_shape() → BasebandModShape

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:SHAPe
value: enums.BasebandModShape = driver.source.bb.general.fm.get_shape()
```

Queries the shape of the modulation signal.

return
fm_shape: SINE

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:[STATe]
value: bool = driver.source.bb.general.fm.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
fm_mod_state: 1| ON| 0| OFF

set_deviation(*fm_deviation: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:DEViation
driver.source.bb.general.fm.set_deviation(fm_deviation = 1.0)
```

Sets the frequency modulation deviation in Hz.

param fm_deviation
float Range: 0 to 4E6

set_frequency(*freq_mod_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:FREQuency
driver.source.bb.general.fm.set_frequency(freq_mod_freq = 1.0)
```

Sets the frequency of the modulation signal.

param freq_mod_freq
float Range: 0.1 to 100E3

set_period(*fm_per: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:PERiod
driver.source.bb.general.fm.set_period(fm_per = 1.0)
```

Queries the period of the modulation signal.

param fm_per
float Range: 100E-9 to 100

set_shape(*fm_shape: BasebandModShape*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:SHAPe
driver.source.bb.general.fm.set_shape(fm_shape = enums.BasebandModShape.SINE)
```

Queries the shape of the modulation signal.

param fm_shape
SINE

set_state(*fm_mod_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:FM:[STATe]
driver.source.bb.general.fm.set_state(fm_mod_state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param fm_mod_state
1| ON| 0| OFF

6.18.3.12.3 Pm

SCPI Commands :

```
[SOURce<HW>]:BB:GENeral:PM:DEViation
[SOURce<HW>]:BB:GENeral:PM:FREQuency
[SOURce<HW>]:BB:GENeral:PM:PERiod
[SOURce<HW>]:BB:GENeral:PM:SHAPe
[SOURce<HW>]:BB:GENeral:PM:[STATe]
```

class PmCls

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

get_deviation() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:DEViation
value: float = driver.source.bb.general.pm.get_deviation()
```

Sets the phase modulation deviation in radians or degrees.

return
pm_deviation: float Range: 0 to 6, Unit: rad

get_frequency() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:FREQuency
value: float = driver.source.bb.general.pm.get_frequency()
```

Sets the frequency of the modulation signal.

return
phase_freq: float Range: 0.1 to 100E3

get_period() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:PERiod
value: float = driver.source.bb.general.pm.get_period()
```

Queries the period of the modulation signal.

return
phase_per: float Range: 100E-9 to 100

get_shape() → BasebandModShape

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:SHAPe
value: enums.BasebandModShape = driver.source.bb.general.pm.get_shape()
```

Queries the shape of the modulation signal.

return
pm_shape: SINE

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:[STATe]
value: bool = driver.source.bb.general.pm.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
phim_mod_state: 1| ON| 0| OFF

set_deviation(*pm_deviation: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:DEViation
driver.source.bb.general.pm.set_deviation(pm_deviation = 1.0)
```

Sets the phase modulation deviation in radians or degrees.

param pm_deviation
float Range: 0 to 6, Unit: rad

set_frequency(*phase_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:FREQuency
driver.source.bb.general.pm.set_frequency(phase_freq = 1.0)
```

Sets the frequency of the modulation signal.

param phase_freq
float Range: 0.1 to 100E3

set_period(*phase_per: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:PERiod
driver.source.bb.general.pm.set_period(phase_per = 1.0)
```

Queries the period of the modulation signal.

param phase_per
float Range: 100E-9 to 100

set_shape(*pm_shape: BasebandModShape*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:SHAPe
driver.source.bb.general.pm.set_shape(pm_shape = enums.BasebandModShape.SINE)
```

Queries the shape of the modulation signal.

param pm_shape
SINE

set_state(*phim_mod_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PM:[STATe]
driver.source.bb.general.pm.set_state(phim_mod_state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param phim_mod_state
1| ON| 0| OFF

6.18.3.12.4 Pulm

SCPI Commands :

```
[SOURce<HW>]:BB:GENeral:PULM:MODE
[SOURce<HW>]:BB:GENeral:PULM:PERiod
[SOURce<HW>]:BB:GENeral:PULM:WIDTh
[SOURce<HW>]:BB:GENeral:PULM:[STATe]
```

class PulmCls

Pulm commands group definition. 8 total commands, 3 Subgroups, 4 group commands

get_mode() → BasebandPulseMode

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:MODE
value: enums.BasebandPulseMode = driver.source.bb.general.pulm.get_mode()
```

Sets the pulse mode. You can set for single or double pulse signals.

return
pulm_mode: SINGle| DOUBle

get_period() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:PERiod
value: float = driver.source.bb.general.pulm.get_period()
```

Defines the pulse period in microseconds.

return
puls_mod_per: float Range: 100E-9 to 100

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:[STATe]
value: bool = driver.source.bb.general.pulm.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
pulm_state: 1| ON| 0| OFF

get_width() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:WIDTh
value: float = driver.source.bb.general.pulm.get_width()
```

Sets the pulse width in microseconds.

return
pulm_width: float Range: 50E-9 to 100

set_mode(*pulm_mode: BasebandPulseMode*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:MODE
driver.source.bb.general.pulm.set_mode(pulm_mode = enums.BasebandPulseMode.
↪DOUBle)
```

Sets the pulse mode. You can set for single or double pulse signals.

param pulm_mode
SINGle| DOUBle

set_period(*puls_mod_per: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:PERiod
driver.source.bb.general.pulm.set_period(puls_mod_per = 1.0)
```

Defines the pulse period in microseconds.

param puls_mod_per
float Range: 100E-9 to 100

set_state(*pulm_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:[STATe]
driver.source.bb.general.pulm.set_state(pulm_state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param pulm_state
1| ON| 0| OFF

set_width(*pulm_width: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:WIDTh
driver.source.bb.general.pulm.set_width(pulm_width = 1.0)
```

Sets the pulse width in microseconds.

param pulm_width
float Range: 50E-9 to 100

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.general.pulm.clone()
```

Subgroups

6.18.3.12.4.1 Double

SCPI Commands :

```
[SOURce<HW>]:BB:GENeral:PULM:DOUBle:DELay
[SOURce<HW>]:BB:GENeral:PULM:DOUBle:WIDTh
```

class DoubleCls

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

get_delay() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:DOUBle:DELay
value: float = driver.source.bb.general.pulm.double.get_delay()
```

Sets the double pulse delay in microseconds.

return
pulm_dbl_del: float Range: 50E-9 to 100

get_width() → float

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:DOUBle:WIDTh
value: float = driver.source.bb.general.pulm.double.get_width()
```

Defines the double pulse width in microseconds.

return
pulm_dbl_width: float Range: 50E-9 to 100

set_delay(*pulm_dbl_del: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:DOUBle:DELay
driver.source.bb.general.pulm.double.set_delay(pulm_dbl_del = 1.0)
```

Sets the double pulse delay in microseconds.

param pulm_dbl_del
float Range: 50E-9 to 100

set_width(*pulm_dbl_width: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:DOUBle:WIDTh
driver.source.bb.general.pulm.double.set_width(pulm_dbl_width = 1.0)
```

Defines the double pulse width in microseconds.

param pulm_dbl_width
float Range: 50E-9 to 100

6.18.3.12.4.2 Transition

SCPI Command :

```
[SOURce<HW>]:BB:GENeral:PULM:TRANsition:TYPE
```

class TransitionCls

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

get_type_py() → BasebandPulseTransType

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:TRANsition:TYPE
value: enums.BasebandPulseTransType = driver.source.bb.general.pulm.transition.
↪get_type_py()
```

Sets the transition type of the pulse modulation signal.

return
pulm_trans_type: FAST| SMOothed

set_type_py(*pulm_trans_type: BasebandPulseTransType*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:TRANsition:TYPE
driver.source.bb.general.pulm.transition.set_type_py(pulm_trans_type = enums.
↪BasebandPulseTransType.FAST)
```

Sets the transition type of the pulse modulation signal.

param pulm_trans_type
FAST| SMOothed

6.18.3.12.4.3 Video

SCPI Command :

```
[SOURce<HW>]:BB:GENeral:PULM:VIDeo:POLarity
```

class VideoCls

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

get_polarity() → NormalInverted

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:VIDeo:POLarity
value: enums.NormalInverted = driver.source.bb.general.pulm.video.get_polarity()
```

Sets the video polarity.

return
puls_video_pol: INVerted| NORMal

set_polarity(*puls_video_pol: NormalInverted*) → None

```
# SCPI: [SOURce<HW>]:BB:GENeral:PULM:VIDeo:POLarity
driver.source.bb.general.pulm.video.set_polarity(puls_video_pol = enums.
↪NormalInverted.INVerted)
```

Sets the video polarity.

param puls_video_pol
INVerted| NORMal

6.18.3.13 Graphics

SCPI Commands :

```
[SOURce]:BB:GRAPhics:ADD
[SOURce]:BB:GRAPhics:CLOSe
[SOURce]:BB:GRAPhics:FFTFscale
[SOURce]:BB:GRAPhics:FFTLen
[SOURce<HW>]:BB:GRAPhics:MODE
[SOURce]:BB:GRAPhics:SOURce
```

class GraphicsCls

Graphics commands group definition. 9 total commands, 2 Subgroups, 6 group commands

close() → None

```
# SCPI: [SOURce]:BB:GRAPhics:CLOSe
driver.source.bb.graphics.close()
```

Closes all graphical signal displays.

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

```
# SCPI: [SOURce]:BB:GRAPhics:CLOSe
driver.source.bb.graphics.close_with_opc()
```

Closes all graphical signal displays.

Same as close, but waits for the operation to complete before continuing further. Use the RsSmcv.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_fft_fscale() → bool

```
# SCPI: [SOURce]:BB:GRAPhics:FFTFscale
value: bool = driver.source.bb.graphics.get_fft_fscale()
```

Defines the normalization of the power values in the power spectrum diagram.

return
state: 1| ON| 0| OFF 1 Normalized power in dBFS 0 Shows power distribution in dB/Hz

get_fft_len() → TranRecFftLen

```
# SCPI: [SOURce]:BB:GRAPhics:FFTLen
value: enums.TranRecFftLen = driver.source.bb.graphics.get_fft_len()
```

Sets the FFT size.

return
mode: LEN256| LEN512| LEN1024| LEN2048| LEN4096

get_mode() → TranRecMode

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:MODE
value: enums.TranRecMode = driver.source.bb.graphics.get_mode()
```

Selects the graphics mode of the graphical signal display.

return
mode: IQ| VECTor| CCDF| PSPectrum| CONStellation| EYEI| EYEQ

get_source() → TranRecSour

```
# SCPI: [SOURce]:BB:GRAPhics:SOURce
value: enums.TranRecSour = driver.source.bb.graphics.get_source()
```

Defines the signal acquisition point, that is the location in the signal flow where the displayed signal is tapped from. See 'Signal acquisition points'.

return
source: STRA| BBA| RFA| BBIA| DO1| IQO1 | STRA| BBA| RFA| BBIA | DO1 STRA Stream A; input stream of the 'IQ Stream Mapper' BBA Baseband signal BBIA Digital baseband input signals RFA RF signal DO1 Digital I/Q output signals; outputs of the 'IQ Stream Mapper'

set_add(*size: TranRecSize*) → None

```
# SCPI: [SOURce]:BB:GRAPhics:ADD
driver.source.bb.graphics.set_add(size = enums.TranRecSize.MAXimized)
```

Adds a graphical signal display (according to the current MODE, SOURce, SRATe:* and TRIGger:* settings) .

param size
MAXimized| MINimized

set_fft_fscale(*state: bool*) → None

```
# SCPI: [SOURce]:BB:GRAPhics:FFTFscale
driver.source.bb.graphics.set_fft_fscale(state = False)
```

Defines the normalization of the power values in the power spectrum diagram.

param state
1| ON| 0| OFF 1 Normalized power in dBFS 0 Shows power distribution in dB/Hz

set_fft_len(*mode: TranRecFftLen*) → None

```
# SCPI: [SOURce]:BB:GRAPhics:FFTLen
driver.source.bb.graphics.set_fft_len(mode = enums.TranRecFftLen.LEN1024)
```

Sets the FFT size.

param mode
LEN256| LEN512| LEN1024| LEN2048| LEN4096

set_mode(*mode: TranRecMode*) → None

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:MODE
driver.source.bb.graphics.set_mode(mode = enums.TranRecMode.CCDF)
```

Selects the graphics mode of the graphical signal display.

param mode
IQ| VECTor| CCDF| PSPectrum| CONStellation| EYEI| EYEQ

set_source(*source: TranRecSour*) → None

```
# SCPI: [SOURce]:BB:GRAPhics:SOURce
driver.source.bb.graphics.set_source(source = enums.TranRecSour.BBA)
```

Defines the signal acquisition point, that is the location in the signal flow where the displayed signal is tapped from. See 'Signal acquisition points'.

param source
STRA| BBA| RFA| BBIA| DO1| IQO1 | STRA| BBA| RFA| BBIA | DO1 STRA Stream A; input stream of the 'IQ Stream Mapper' BBA Baseband signal BBIA Digital baseband input signals RFA RF signal DO1 Digital I/Q output signals; outputs of the 'IQ Stream Mapper'

Cloning the Group

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

Subgroups

6.18.3.13.1 SymbolRate

SCPI Commands :

```
[SOURce<HW>]:BB:GRAPhics:SRATe:MODE
[SOURce<HW>]:BB:GRAPhics:SRATe:USER
```

class SymbolRateCls

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

get_mode() → TranRecSampFactMode

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:SRATe:MODE
value: enums.TranRecSampFactMode = driver.source.bb.graphics.symbolRate.get_
↪mode()
```

Sets how the time resolution of the signal is determined. Maximum resolution corresponds to a diagram covering the entire signal bandwidth. The higher the resolution is, the shorter the length of the displayed signal segment will be for the specified recording depth.

return
mode: AUTO| FULL| USER

get_user() → float

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:SRATe:USER
value: float = driver.source.bb.graphics.symbolRate.get_user()
```

(Enabled for BB:GRAPH:SRAT:MODE USER) Selects the signal bandwidth for the diagram. The setting range moves between the minimum and maximum bandwidth which is possible for the selected graphical signal display. The selection is made graphically by moving the pointer.

return
user: float Range: 0.01 to 100, Unit: PCT

set_mode(*mode: TranRecSampFactMode*) → None

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:SRATe:MODE
driver.source.bb.graphics.symbolRate.set_mode(mode = enums.TranRecSampFactMode.
↪AUTO)
```

Sets how the time resolution of the signal is determined. Maximum resolution corresponds to a diagram covering the entire signal bandwidth. The higher the resolution is, the shorter the length of the displayed signal segment will be for the specified recording depth.

param mode
AUTO| FULL| USER

set_user(*user: float*) → None

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:SRATe:USER
driver.source.bb.graphics.symbolRate.set_user(user = 1.0)
```

(Enabled for BB:GRAPH:SRAT:MODE USER) Selects the signal bandwidth for the diagram. The setting range moves between the minimum and maximum bandwidth which is possible for the selected graphical signal display. The selection is made graphically by moving the pointer.

param user
float Range: 0.01 to 100, Unit: PCT

6.18.3.13.2 Trigger

SCPI Command :

```
[SOURce<HW>]:BB:GRAPhics:TRIGger:SOURce
```

class TriggerCls

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

get_source() → TranRecTrigSour

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:TRIGger:SOURce
value: enums.TranRecTrigSour = driver.source.bb.graphics.trigger.get_source()
```

Defines the trigger for the starting time of the graphic recording.

return
source: SOFTware| MARKer

set_source(*source: TranRecTrigSour*) → None

```
# SCPI: [SOURce<HW>]:BB:GRAPhics:TRIGger:SOURce
driver.source.bb.graphics.trigger.set_source(source = enums.TranRecTrigSour.
↪MARKer)
```

Defines the trigger for the starting time of the graphic recording.

param source
SOFTware| MARKer

6.18.3.14 Impairment

SCPI Commands :

```
[SOURce<HW>]:BB:IMPairment:DELay
[SOURce<HW>]:BB:IMPairment:STATe
```

class ImpairmentCls

Impairment commands group definition. 16 total commands, 5 Subgroups, 2 group commands

get_delay() → float

```
# SCPI: [SOURce<HW>]:BB:IMPairment:DELay
value: float = driver.source.bb.impairment.get_delay()
```

No command help available

return
delay: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:IMPairment:STATe
value: bool = driver.source.bb.impairment.get_state()
```

Activates the impairment or correction values LEAKage, QUADrature and IQRatio.

return
state: 1| ON| 0| OFF

set_delay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:DELay
driver.source.bb.impairment.set_delay(delay = 1.0)
```

No command help available

param delay
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:STATe
driver.source.bb.impairment.set_state(state = False)
```

Activates the impairment or correction values LEAKage, QUADrature and IQRatio.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.18.3.14.1 IqOutput<IqConnector>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.source.bb.impairment.iqOutput.repcap_iqConnector_get()
driver.source.bb.impairment.iqOutput.repcap_iqConnector_set(repcap.IqConnector.Nr1)
```

class IqOutputCls

IqOutput commands group definition. 8 total commands, 7 Subgroups, 0 group commands Repeated Capability: IqConnector, default value after init: IqConnector.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.impairment.iqOutput.clone()
```

Subgroups

6.18.3.14.1.1 Delay

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:DELay
```

class DelayCls

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

get(*iqConnector=IqConnector.Default*) → float

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:DELay
value: float = driver.source.bb.impairment.iqOutput.delay.get(iqConnector =␣
↪repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
delay: No help available

set(*delay: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:DELay
driver.source.bb.impairment.iqOutput.delay.set(delay = 1.0, iqConnector =␣
↪repcap.IqConnector.Default)
```

No command help available

param delay
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.1.2 IqRatio

class IqRatioCls

IqRatio 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.bb.impairment.iqOutput.iqRatio.clone()
```

Subgroups

6.18.3.14.1.3 Magnitude

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:IQRatio:[MAGNitude]
```

class MagnitudeCls

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

get(*iqConnector=IqConnector.Default*) → float

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:IQRatio:[MAGNitude]
value: float = driver.source.bb.impairment.iqOutput.iqRatio.magnitude.
↪get(iqConnector = repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
ipartq_ratio: No help available

set(*ipartq_ratio: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:IQRatio:[MAGNitude]
driver.source.bb.impairment.iqOutput.iqRatio.magnitude.set(ipartq_ratio = 1.0,␣
↪iqConnector = repcap.IqConnector.Default)
```

No command help available

param ipartq_ratio
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.1.4 Leakage

class LeakageCls

Leakage 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.bb.impairment.iqOutput.leakage.clone()
```

Subgroups

6.18.3.14.1.5 Icomponent

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:LEAKage:I
```

class IcomponentCls

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

get(*iqConnector=IqConnector.Default*) → float

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:LEAKage:I
value: float = driver.source.bb.impairment.iqOutput.leakage.icomponent.
↪get(iqConnector = repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
ipart: No help available

set(*ipart: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:LEAKage:I
driver.source.bb.impairment.iqOutput.leakage.icomponent.set(ipart = 1.0,␣
↪iqConnector = repcap.IqConnector.Default)
```

No command help available

param ipart
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.1.6 Qcomponent

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:LEAKage:Q
```

class QcomponentCls

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

get(*iqConnector=IqConnector.Default*) → float

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:LEAKage:Q
value: float = driver.source.bb.impairment.iqOutput.leakage.qcomponent.
↪get(iqConnector = repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
qpart: No help available

set(*qpart: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:LEAKage:Q
driver.source.bb.impairment.iqOutput.leakage.qcomponent.set(qpart = 1.0,␣
↪iqConnector = repcap.IqConnector.Default)
```

No command help available

param qpart
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.1.7 Poffset

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:POFFset
```

class PoffsetCls

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

get(*iqConnector=IqConnector.Default*) → float

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:POFFset
value: float = driver.source.bb.impairment.iqOutput.poffset.get(iqConnector =␣
↪repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
phase_offset: No help available

set(*phase_offset: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:POFFset
driver.source.bb.impairment.iqOutput.poffset.set(phase_offset = 1.0,␣
↪iqConnector = repcap.IqConnector.Default)
```

No command help available

param phase_offset
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.1.8 Quadrature

class QuadratureCls

Quadrature 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.bb.impairment.iqOutput.quadrature.clone()
```

Subgroups

6.18.3.14.1.9 Angle

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:QUADrature:[ANGLe]
```

class AngleCls

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

get(*iqConnector=IqConnector.Default*) → float

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:QUADrature:[ANGLe]
value: float = driver.source.bb.impairment.iqOutput.quadrature.angle.
↪get(iqConnector = repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
angle: No help available

set(*angle: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:QUADrature:[ANGLe]
driver.source.bb.impairment.iqOutput.quadrature.angle.set(angle = 1.0,␣
↪iqConnector = repcap.IqConnector.Default)
```

No command help available

param angle
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.1.10 Skew

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:SKEW
```

class SkewCls

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

get(*iqConnector=IqConnector.Default*) → float

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:SKEW
value: float = driver.source.bb.impairment.iqOutput.skew.get(iqConnector =␣
↪repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
skew: No help available

set(*skew: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:SKEW
driver.source.bb.impairment.iqOutput.skew.set(skew = 1.0, iqConnector = repcap.
↪IqConnector.Default)
```

No command help available

param skew
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.1.11 State

SCPI Command :

```
[SOURce]:BB:IMPairment:IQOutput<CH>:STATe
```

class StateCls

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

get(*iqConnector=IqConnector.Default*) → bool

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:STATe
value: bool = driver.source.bb.impairment.iqOutput.state.get(iqConnector =␣
↪repcap.IqConnector.Default)
```

No command help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

return
state: No help available

set(*state: bool*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:IQOutput<CH>:STATe
driver.source.bb.impairment.iqOutput.state.set(state = False, iqConnector =␣
↪repcap.IqConnector.Default)
```

No command help available

param state
No help available

param iqConnector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'IqOutput')

6.18.3.14.2 IqRatio

SCPI Command :

```
[SOURce<HW>]:BB:IMPairment:IQRatio:[MAGNitude]
```

class IqRatioCls

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

get_magnitude() → float

```
# SCPI: [SOURce<HW>]:BB:IMPairment:IQRatio:[MAGNitude]
value: float = driver.source.bb.impairment.iqRatio.get_magnitude()
```

No command help available

return
ipartq_ratio: No help available

set_magnitude(*ipartq_ratio: float*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:IQRatio:[MAGNitude]
driver.source.bb.impairment.iqRatio.set_magnitude(ipartq_ratio = 1.0)
```

No command help available

param ipartq_ratio
float Range: -1 to 1

6.18.3.14.3 Leakage

SCPI Commands :

```
[SOURce<HW>]:BB:IMPairment:LEAKage:I
[SOURce<HW>]:BB:IMPairment:LEAKage:Q
```

class LeakageCls

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

get_icomponent() → float

```
# SCPI: [SOURce<HW>]:BB:IMPairment:LEAKage:I
value: float = driver.source.bb.impairment.leakage.get_icomponent()
```

Determines the leakage amplitude of the I or Q signal component of the corresponding stream

return
ipart: No help available

get_qcomponent() → float

```
# SCPI: [SOURce<HW>]:BB:IMPairment:LEAKage:Q
value: float = driver.source.bb.impairment.leakage.get_qcomponent()
```

Determines the leakage amplitude of the I or Q signal component of the corresponding stream

return
qpart: No help available

set_icomponent(*ipart: float*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:LEAKage:I
driver.source.bb.impairment.leakage.set_icomponent(ipart = 1.0)
```

Determines the leakage amplitude of the I or Q signal component of the corresponding stream

param ipart
float Range: -10 to 10

set_qcomponent(*qpart: float*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:LEAKage:Q
driver.source.bb.impairment.leakage.set_qcomponent(qpart = 1.0)
```

Determines the leakage amplitude of the I or Q signal component of the corresponding stream

param qpart
float Range: -10 to 10

6.18.3.14.4 Optimization

SCPI Commands :

```
[SOURce<HW>]:BB:IMPairment:OPTimization:MODE
[SOURce<HW>]:BB:IMPairment:OPTimization:STATe
```

class OptimizationCls

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

get_mode() → BbImpOptMode

```
# SCPI: [SOURce<HW>]:BB:IMPairment:OPTimization:MODE
value: enums.BbImpOptMode = driver.source.bb.impairment.optimization.get_mode()
```

Sets the optimization mode.

return
mode: FAST| | QHTable FAST Optimization by compensation for I/Q skew. QHTable Improved optimization by maintained speed.

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:IMPairment:OPTimization:STATe
value: bool = driver.source.bb.impairment.optimization.get_state()
```

No command help available

return
state: No help available

set_mode(*mode: BbImpOptMode*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:OPTimization:MODE
driver.source.bb.impairment.optimization.set_mode(mode = enums.BbImpOptMode.
↪FAST)
```

Sets the optimization mode.

param mode
FAST| | QHTable FAST Optimization by compensation for I/Q skew. QHTable Improved optimization by maintained speed.

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:OPTimization:STATe
driver.source.bb.impairment.optimization.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.14.5 Quadrature

SCPI Command :

```
[SOURce<HW>]:BB:IMPairment:QUADrature:[ANGLe]
```

class QuadratureCls

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

get_angle() → float

```
# SCPI: [SOURce<HW>]:BB:IMPairment:QUADrature:[ANGLe]
value: float = driver.source.bb.impairment.quadrature.get_angle()
```

Sets a quadrature offset (phase angle) between the I and Q vectors deviating from the ideal 90 degrees. A positive quadrature offset results in a phase angle greater than 90 degrees.

return
angle: float Range: -10 to 10, Unit: DEG

set_angle(*angle: float*) → None

```
# SCPI: [SOURce<HW>]:BB:IMPairment:QUADrature:[ANGLe]
driver.source.bb.impairment.quadrature.set_angle(angle = 1.0)
```

Sets a quadrature offset (phase angle) between the I and Q vectors deviating from the ideal 90 degrees. A positive quadrature offset results in a phase angle greater than 90 degrees.

param angle
float Range: -10 to 10, Unit: DEG

6.18.3.15 Info

SCPI Command :

```
[SOURce]:BB:INFO:PSEQuencer
```

class InfoCls

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

get_psequencer() → str

```
# SCPI: [SOURce]:BB:INFO:PSEQuencer
value: str = driver.source.bb.info.get_psequencer()
```

No command help available

return
info_xml_string: No help available

6.18.3.16 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:INPut:FORMat
[SOURce<HW>]:BB:INPut:TSCHannel
[SOURce<HW>]:BB:INPut
```

class InputPyCls

InputPy commands group definition. 11 total commands, 1 Subgroups, 3 group commands

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.inputPy.get_format_py()
```

No command help available

return
input_format: No help available

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.inputPy.get_ts_channel()
```

No command help available

return
ts_channel: No help available

get_value() → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:INPut
value: enums.CodingInputSignalInputA = driver.source.bb.inputPy.get_value()
```

No command help available

return
input_py: No help available

set_format_py(*input_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:FORMat
driver.source.bb.inputPy.set_format_py(input_format = enums.CodingInputFormat.
↪ASI)
```

No command help available

param input_format
No help available

set_ts_channel(*ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:TSCHannel
driver.source.bb.inputPy.set_ts_channel(ts_channel = enums.NumberA._1)
```

No command help available

param ts_channel
No help available

set_value(*input_py: CodingInputSignalInputA*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut
driver.source.bb.inputPy.set_value(input_py = enums.CodingInputSignalInputA.
↪ASI1)
```

No command help available

param input_py
No help available

Cloning the Group

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

Subgroups

6.18.3.16.1 Ip<IpVersion>

RepCap Settings

```
# Range: Nr4 .. Nr6
rc = driver.source.bb.inputPy.ip.repcap_ipVersion_get()
driver.source.bb.inputPy.ip.repcap_ipVersion_set(repcap.IpVersion.Nr4)
```

class IpCls

Ip commands group definition. 8 total commands, 6 Subgroups, 0 group commands Repeated Capability: IpVersion, default value after init: IpVersion.Nr4

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.inputPy.ip.clone()
```

Subgroups

6.18.3.16.1.1 Alias

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:ALIas
```

class AliasCls

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

get(*ipVersion=IpVersion.Default*) → str

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:ALIas
value: str = driver.source.bb.inputPy.ip.alias.get(ipVersion = repcap.IpVersion.
↪Default)
```

Specifies an alias, i.e. name for the IP connection.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

return
alias: string

set(*alias: str*, *ipVersion=IpVersion.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:ALIas
driver.source.bb.inputPy.ip.alias.set(alias = 'abc', ipVersion = repcap.
↪IpVersion.Default)
```

Specifies an alias, i.e. name for the IP connection.

param alias
string

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

6.18.3.16.1.2 Igmp

class IgmpCls

Igmp 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.source.bb.inputPy.ip.igmp.clone()
```

Subgroups

6.18.3.16.1.3 Source

class SourceCls

Source 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.bb.inputPy.ip.igmp.source.clone()
```

Subgroups

6.18.3.16.1.4 Address

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:IGMP:[SOURce]:ADDRess
```

class AddressCls

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

get(*ipVersion=IpVersion.Default*) → bytes

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:IGMP:[SOURce]:ADDRess
value: bytes = driver.source.bb.inputPy.ip.igmp.source.address.get(ipVersion =␣
↪repcap.IpVersion.Default)
```

Specifies the IGMP source address of the network.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

return
bc_coding_ip_igmpv_3_source_address: No help available

set(*bc_coding_ip_igmpv_3_source_address: bytes*, *ipVersion=IpVersion.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:IGMP:[SOURce]:ADDRess
driver.source.bb.inputPy.ip.igmp.source.address.set(bc_coding_ip_igmpv_3_source_
↪address = b'ABCDEFGH', ipVersion = repcap.IpVersion.Default)
```

Specifies the IGMP source address of the network.

param bc_coding_ip_igmpv_3_source_address
string

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

6.18.3.16.1.5 Ping

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:IGMP:[SOURce]:PING
```

class PingCls

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

set(*ipVersion=IpVersion.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:IGMP:[SOURce]:PING
driver.source.bb.inputPy.ip.igmp.source.ping.set(ipVersion = repcap.IpVersion.
↪Default)
```

Triggers pinging of the IGMP source address in the local IP data network. Query the result via [:SOURce<hw>]:BB:INPut:IP<ch>:IGMP[:SOURce]:RESult?.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

set_with_opc(*ipVersion=IpVersion.Default*, *opc_timeout_ms: int = -1*) → None

6.18.3.16.1.6 Result

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:IGMP:[SOURce]:RESult
```

class ResultCls

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

get(*ipVersion=IpVersion.Default*) → str

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:IGMP:[SOURce]:RESult
value: str = driver.source.bb.inputPy.ip.igmp.source.result.get(ipVersion =␣
↪repcap.IpVersion.Default)
```

Queries the result of pinging the source address. See [:SOURce<hw>]:BB:INPut:IP<ch>:IGMP[:SOURce]:PING.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

return
ping_result: string Returns ping messages.

6.18.3.16.1.7 Multicast

class MulticastCls

Multicast 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.bb.inputPy.ip.multicast.clone()
```

Subgroups

6.18.3.16.1.8 Address

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:MULticast:ADDRess
```

class AddressCls

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

get(*ipVersion=IpVersion.Default*) → bytes

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:MULticast:ADDRess
value: bytes = driver.source.bb.inputPy.ip.multicast.address.get(ipVersion =␣
↪repcap.IpVersion.Default)
```

Sets the destination IP address (IPv4) of the IP connection.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

return
bc_coding_ip_multicast_address: No help available

set(*bc_coding_ip_multicast_address: bytes*, *ipVersion=IpVersion.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:MULticast:ADDRess
driver.source.bb.inputPy.ip.multicast.address.set(bc_coding_ip_multicast_
↪address = b'ABCDEFGH', ipVersion = repcap.IpVersion.Default)
```

Sets the destination IP address (IPv4) of the IP connection.

param bc_coding_ip_multicast_address
string Range: 224.0.0.0 to 239.255.255.255

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

6.18.3.16.1.9 Port

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:PORT
```

class PortCls

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

get(*ipVersion=IpVersion.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:PORT
value: int = driver.source.bb.inputPy.ip.port.get(ipVersion = repcap.IpVersion.
↪Default)
```

Sets the port of the input IP data at the 'IP Data' connector.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

return
port: integer Range: 0 to 65535

set(*port: int*, *ipVersion=IpVersion.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:PORT
driver.source.bb.inputPy.ip.port.set(port = 1, ipVersion = repcap.IpVersion.
↪Default)
```

Sets the port of the input IP data at the 'IP Data' connector.

param port
integer Range: 0 to 65535

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

6.18.3.16.1.10 State

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:[STATe]
```

class StateCls

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

get(*ipVersion=IpVersion.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:[STATe]
value: bool = driver.source.bb.inputPy.ip.state.get(ipVersion = repcap.
↪IpVersion.Default)
```

Activates/deactivates the 'IP Channel x' as IP input. Specify the current IP TS Channel with the command SOURce1:BB:DigStd:INPut:TSCHannel. DigStd stands for the IP TS Channel in the corresponding broadcast standard.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

return
alias: 1| ON| 0| OFF

set(*alias: bool*, *ipVersion=IpVersion.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:[STATe]
driver.source.bb.inputPy.ip.state.set(alias = False, ipVersion = repcap.
↪IpVersion.Default)
```

Activates/deactivates the 'IP Channel x' as IP input. Specify the current IP TS Channel with the command SOURce1:BB:DigStd:INPut:TSCHannel. DigStd stands for the IP TS Channel in the corresponding broadcast standard.

param alias
1| ON| 0| OFF

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

6.18.3.16.1.11 TypePy

SCPI Command :

```
[SOURce<HW>]:BB:INPut:IP<CH>:TYPE
```

class TypePyCls

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

get(*ipVersion=IpVersion.Default*) → CodingIpType

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:TYPE
value: enums.CodingIpType = driver.source.bb.inputPy.ip.typePy.get(ipVersion =␣
↪repcap.IpVersion.Default)
```

Sets the IP input type.

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

return
type_py: UNIcast| MULTicast UNIcast Analyzes all unicast IP packets that arrive at the specified port. See [:SOURcehw]:BB:INPut:IPch:PORT. MULTicast When an IP address is in the multicast address range, an attempt is made to join a multicast group using . Set multicast address and port. See: [:SOURcehw]:BB:INPut:IPch:MULticast:ADDRess [:SOURcehw]:BB:INPut:IPch:PORT

set(*type_py: CodingIpType*, *ipVersion=IpVersion.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:INPut:IP<CH>:TYPE
driver.source.bb.inputPy.ip.typePy.set(type_py = enums.CodingIpType.MULTicast,␣
↪ipVersion = repcap.IpVersion.Default)
```

Sets the IP input type.

param type_py
UNIcast| MULTicast UNIcast Analyzes all unicast IP packets that arrive at the specified port. See [:SOURcehw]:BB:INPut:IPch:PORT. MULTicast When an IP address is in the multicast address range, an attempt is made to join a multicast group using . Set multicast address and port. See: [:SOURcehw]:BB:INPut:IPch:MULticast:ADDRess [:SOURcehw]:BB:INPut:IPch:PORT

param ipVersion
optional repeated capability selector. Default value: Nr4 (settable in the interface 'Ip')

6.18.3.17 Isdbt

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:BANDwidth
[SOURce<HW>]:BB:ISDBt:CONTrol
[SOURce<HW>]:BB:ISDBt:GUARd
[SOURce<HW>]:BB:ISDBt:NETWorkmode
[SOURce<HW>]:BB:ISDBt:PACKetlength
[SOURce<HW>]:BB:ISDBt:PID
[SOURce<HW>]:BB:ISDBt:PIDTestpack
[SOURce<HW>]:BB:ISDBt:PORTion
[SOURce<HW>]:BB:ISDBt:PRESet
[SOURce<HW>]:BB:ISDBt:REMux
[SOURce<HW>]:BB:ISDBt:STATe
[SOURce<HW>]:BB:ISDBt:STUFfing
[SOURce<HW>]:BB:ISDBt:SUBChannel
[SOURce<HW>]:BB:ISDBt:SYSTem
```

class IsdbtCls

Isdbt commands group definition. 71 total commands, 17 Subgroups, 14 group commands

get_bandwidth() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:BANDwidth
value: int = driver.source.bb.isdbt.get_bandwidth()
```

Displays the used bandwidth.

return
used_bw: integer Range: 0 to 9999

get_control() → AutoManualMode

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONTrol
value: enums.AutoManualMode = driver.source.bb.isdbt.get_control()
```

Defines the configuration mode of the coder.

return
control: AUTO| MANual

get_guard() → CodingGuardInterval

```
# SCPI: [SOURce<HW>]:BB:ISDBt:GUARd
value: enums.CodingGuardInterval = driver.source.bb.isdbt.get_guard()
```

Sets the guard interval length.

return
guard_int: G1_32| G1_16| G1_8| G1_4

get_network_mode() → NetworkMode

```
# SCPI: [SOURce<HW>]:BB:ISDBt:NETWorkmode
value: enums.NetworkMode = driver.source.bb.isdbt.get_network_mode()
```

No command help available

return
network_mode: No help available

get_packet_length() → InputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PACKetlength
value: enums.InputSignalPacketLength = driver.source.bb.isdbt.get_packet_
↪length()
```

Queries the packet length of the external transport stream in bytes.

return
packet_length: P188| P204| INValid P188|P204 188/204 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PID
value: int = driver.source.bb.isdbt.get_pid()
```

Sets the .

return
pid: integer Range: #H000 to #H1FFF

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.isdbt.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
test_pack: VARiable| NULL

get_portion() → CodingPortions

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PORTion
value: enums.CodingPortions = driver.source.bb.isdbt.get_portion()
```

Sets the modulation types of the hierachical layers A, B and C. The first digit specifies the modulation type for layer A, the second digit for layer B and the third digit for layer C.

return
portion: PDD| PDC| PCC| DDD| DDC| DCC| CCC P Partial reception D Differential modulation C Coherent modulation

get_remux() → bool

```
# SCPI: [SOURce<HW>]:BB:ISDBt:REMux
value: bool = driver.source.bb.isdbt.get_remux()
```

Enables/disables the built-in remultiplexer.

return
remux: 1| ON| 0| OFF

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ISDBt:STATe
value: bool = driver.source.bb.isdbt.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_stuffing() → bool

```
# SCPI: [SOURce<HW>]:BB:ISDBt:STUFfing
value: bool = driver.source.bb.isdbt.get_stuffing()
```

Queries, if stuffing is enabled or disabled.
INTRO_CMD_HELP: You can enable/disable stuffing via [:SOURce<hw>]:BB:ISDBt:CONTrol:

- SOURce1:BB:ISDBt:CONTrol AUTO Stuffing is disabed.
- SOURce1:BB:ISDBt:CONTrol MAN Stuffing is enabed.

return
stuffing: 1| ON| 0| OFF

get_sub_channel() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SUBChannel
value: int = driver.source.bb.isdbt.get_sub_channel()
```

Sets the subchannel of the ISDB-TSB signal.

return
sub_channel: integer Range: 0 to 41

get_system() → IsdbtCodingSystem

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SYSTem
value: enums.IsdbtCodingSystem = driver.source.bb.isdbt.get_system()
```

Sets the ISDB-T system.

return
system: TSB3| TSB1| T

preset() → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PRESet
driver.source.bb.isdbt.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>:BB:ISDBt:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PRESet
driver.source.bb.isdbt.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>:BB:ISDBt:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_control(*control: AutoManualMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONTrol
driver.source.bb.isdbt.set_control(control = enums.AutoManualMode.AUTO)
```

Defines the configuration mode of the coder.

param control
AUTO| MANual

set_guard(*guard_int: CodingGuardInterval*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:GUARd
driver.source.bb.isdbt.set_guard(guard_int = enums.CodingGuardInterval.G1_16)
```

Sets the guard interval length.

param guard_int
G1_32| G1_16| G1_8| G1_4

set_network_mode(*network_mode: NetworkMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:NETWorkmode
driver.source.bb.isdbt.set_network_mode(network_mode = enums.NetworkMode.MFN)
```

No command help available

param network_mode
No help available

set_pid(*pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PID
driver.source.bb.isdbt.set_pid(pid = 1)
```

Sets the .

param pid
integer Range: #H000 to #H1FFF

set_pid_test_pack(*test_pack: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PIDTestpack
driver.source.bb.isdbt.set_pid_test_pack(test_pack = enums.PidTestPacket.NULL)
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

param test_pack
VARiable| NULL

set_portion(*portion: CodingPortions*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PORTion
driver.source.bb.isdbt.set_portion(portion = enums.CodingPortions.CCC)
```

Sets the modulation types of the hierachical layers A, B and C. The first digit specifies the modulation type for layer A, the second digit for layer B and the third digit for layer C.

param portion
PDD| PDC| PCC| DDD| DDC| DCC| CCC P Partial reception D Differential modulation C Coherent modulation

set_remux(*remux: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:REMux
driver.source.bb.isdbt.set_remux(remux = False)
```

Enables/disables the built-in remultiplexer.

param remux
1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:STATe
driver.source.bb.isdbt.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

set_sub_channel(*sub_channel: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SUBChannel
driver.source.bb.isdbt.set_sub_channel(sub_channel = 1)
```

Sets the subchannel of the ISDB-TSB signal.

param sub_channel
integer Range: 0 to 41

set_system(*system: IsdbtCodingSystem*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SYSTem
driver.source.bb.isdbt.set_system(system = enums.IsdbtCodingSystem.T)
```

Sets the ISDB-T system.

param system
TSB3| TSB1| T

Cloning the Group

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

Subgroups

6.18.3.17.1 Channel

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:CHANnel:[BANDwidth]
```

class ChannelCls

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

get_bandwidth() → CodingChannelBandwidth

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CHANnel:[BANDwidth]
value: enums.CodingChannelBandwidth = driver.source.bb.isdbt.channel.get_
↪bandwidth()
```

Selects the channel bandwidth.

return
channel_bw: BW_8| BW_6| BW_7

set_bandwidth(*channel_bw: CodingChannelBandwidth*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CHANnel:[BANDwidth]
driver.source.bb.isdbt.channel.set_bandwidth(channel_bw = enums.
↪CodingChannelBandwidth.BW_6)
```

Selects the channel bandwidth.

param channel_bw
BW_8| BW_6| BW_7

6.18.3.17.2 Constel

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:CONStel:A
[SOURce<HW>]:BB:ISDBt:CONStel:B
[SOURce<HW>]:BB:ISDBt:CONStel:C
```

class ConstelCls

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

get_a() → CodingIsdbtCodingConstel

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONStel:A
value: enums.CodingIsdbtCodingConstel = driver.source.bb.isdbt.constel.get_a()
```

Defines the constellation.

return
constel_a: No help available

get_b() → CodingIsdbtCodingConstel

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONStel:B
value: enums.CodingIsdbtCodingConstel = driver.source.bb.isdbt.constel.get_b()
```

Defines the constellation.

return
constel_b: No help available

get_c() → CodingIsdbtCodingConstel

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONStel:C
value: enums.CodingIsdbtCodingConstel = driver.source.bb.isdbt.constel.get_c()
```

Defines the constellation.

return
constel_c: C_DQPSK| C_QPSK| C_16QAM| C_64QAM

set_a(*constel_a: CodingIsdbtCodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONStel:A
driver.source.bb.isdbt.constel.set_a(constel_a = enums.CodingIsdbtCodingConstel.
↪C_16QAM)
```

Defines the constellation.

param constel_a
C_DQPSK| C_QPSK| C_16QAM| C_64QAM

set_b(*constel_b: CodingIsdbtCodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONStel:B
driver.source.bb.isdbt.constel.set_b(constel_b = enums.CodingIsdbtCodingConstel.
↪C_16QAM)
```

Defines the constellation.

param constel_b
C_DQPSK| C_QPSK| C_16QAM| C_64QAM

set_c(*constel_c: CodingIsdbtCodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:CONStel:C
driver.source.bb.isdbt.constel.set_c(constel_c = enums.CodingIsdbtCodingConstel.
↪C_16QAM)
```

Defines the constellation.

param constel_c
C_DQPSK| C_QPSK| C_16QAM| C_64QAM

6.18.3.17.3 Eew

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:EEW:AREAinfo
[SOURce<HW>]:BB:ISDBt:EEW:EEW
[SOURce<HW>]:BB:ISDBt:EEW:NUMepicenter
[SOURce<HW>]:BB:ISDBt:EEW:SIGNaltype
```

class EewCls

Eew commands group definition. 13 total commands, 9 Subgroups, 4 group commands

get_area_info() → List[str]

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:AREAinfo
value: List[str] = driver.source.bb.isdbt.eew.get_area_info()
```

Sets the target area of the seismic motion warning in hexadecimal presentation.

return
area_inf: integer Range: #H00000000000000 to #HFFFFFFFFFFFFFF

get_eew() → bool

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:EEW
value: bool = driver.source.bb.isdbt.eew.get_eew()
```

Enables/disables the system.

return
eew: 1| ON| 0| OFF

get_num_epicenter() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:NUMepicenter
value: int = driver.source.bb.isdbt.eew.get_num_epicenter()
```

Identifies the total number of seismic motion information being transmitted.

return
num_epicenter: integer Range: 1 to 2

get_signal_type() → IsdbtEewSignalType

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:SIGNaltype
value: enums.IsdbtEewSignalType = driver.source.bb.isdbt.eew.get_signal_type()
```

Identifies the type of seismic motion warning.

return
signal_type: WWA| WWOA| TWA| TWOA

set_area_info(*area_inf: List[str]*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:AREAinfo
driver.source.bb.isdbt.eew.set_area_info(area_inf = ['rawAbc1', 'rawAbc2',
↪'rawAbc3'])
```

Sets the target area of the seismic motion warning in hexadecimal presentation.

param area_inf
integer Range: #H00000000000000 to #HFFFFFFFFFFFFFF

set_eew(*eew: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:EEW
driver.source.bb.isdbt.eew.set_eew(eew = False)
```

Enables/disables the system.

param eew
1| ON| 0| OFF

set_num_epicenter(*num_epicenter: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:NUMepicenter
driver.source.bb.isdbt.eew.set_num_epicenter(num_epicenter = 1)
```

Identifies the total number of seismic motion information being transmitted.

param num_epicenter
integer Range: 1 to 2

set_signal_type(*signal_type: IsdbtEewSignalType*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:SIGNaltype
driver.source.bb.isdbt.eew.set_signal_type(signal_type = enums.
↪IsdbtEewSignalType.TWA)
```

Identifies the type of seismic motion warning.

param signal_type
WWA| WWOA| TWA| TWOA

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.eew.clone()
```

Subgroups

6.18.3.17.3.1 Apai

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:APAI
```

class ApaiCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:APAI
driver.source.bb.isdbt.eew.apai.set()
```

Issues a seismic motion warning based on the information of 'Area Information Hex'.

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

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:APAI
driver.source.bb.isdbt.eew.apai.set_with_opc()
```

Issues a seismic motion warning based on the information of 'Area Information Hex'.

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

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

6.18.3.17.3.2 Ape1

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:APE1
```

class Ape1Cls

Ape1 commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:APE1
driver.source.bb.isdbt.eew.ape1.set()
```

Issues a seismic motion warning based on the settings for epicenter 1.

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

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:APE1
driver.source.bb.isdbt.eew.ape1.set_with_opc()
```

Issues a seismic motion warning based on the settings for epicenter 1.

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

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

6.18.3.17.3.3 Ape2

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:APE2
```

class Ape2Cls

Ape2 commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:APE2
driver.source.bb.isdbt.eew.ape2.set()
```

Issues a seismic motion warning based on the settings for epicenter 2.

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

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:APE2
driver.source.bb.isdbt.eew.ape2.set_with_opc()
```

Issues a seismic motion warning based on the settings for epicenter 2.

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

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

6.18.3.17.3.4 Depth<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.isdbt.eew.depth.repcap_index_get()
driver.source.bb.isdbt.eew.depth.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:DEPTh<CH>
```

class DepthCls

Depth commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:DEPTh<CH>
value: float = driver.source.bb.isdbt.eew.depth.get(index = repcap.Index.
↪Default)
```

Sets the depth of the epicenter of the seismic event.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Depth')

return
depth: float Range: 0 M to 1023 M

set(*depth: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:DEPTh<CH>
driver.source.bb.isdbt.eew.depth.set(depth = 1.0, index = repcap.Index.Default)
```

Sets the depth of the epicenter of the seismic event.

param depth
float Range: 0 M to 1023 M

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Depth')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.eew.depth.clone()
```

6.18.3.17.3.5 InfoType<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.isdbt.eew.infoType.repcap_index_get()
driver.source.bb.isdbt.eew.infoType.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:INFotype<CH>
```

class InfoTypeCls

InfoType commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → IsdbtEewInfoType

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:INFotype<CH>
value: enums.IsdbtEewInfoType = driver.source.bb.isdbt.eew.infoType.get(index =␣
↪repcap.Index.Default)
```

Provides information about the validity of the seismic motion warning.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InfoType')

return
info_type: CANCeled| ISSued

set(*info_type: IsdbtEewInfoType*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:INFotype<CH>
driver.source.bb.isdbt.eew.infoType.set(info_type = enums.IsdbtEewInfoType.
↪CANCeled, index = repcap.Index.Default)
```

Provides information about the validity of the seismic motion warning.

param info_type
CANCeled| ISSued

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InfoType')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.eew.infoType.clone()
```

6.18.3.17.3.6 Latitude<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.isdbt.eew.latitude.repcap_index_get()
driver.source.bb.isdbt.eew.latitude.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:LATitude<CH>
```

class LatitudeCls

Latitude commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:LATitude<CH>
value: float = driver.source.bb.isdbt.eew.latitude.get(index = repcap.Index.
↪Default)
```

Sets the geographical latitude of the epicenter of the seismic event.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Latitude')

return
latitude: float Range: -90.0 DEG to 90 DEG

set(*latitude: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:LATitude<CH>
driver.source.bb.isdbt.eew.latitude.set(latitude = 1.0, index = repcap.Index.
↪Default)
```

Sets the geographical latitude of the epicenter of the seismic event.

param latitude
float Range: -90.0 DEG to 90 DEG

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Latitude')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.eew.latitude.clone()
```

6.18.3.17.3.7 Longitude<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.isdbt.eew.longitude.repcap_index_get()
driver.source.bb.isdbt.eew.longitude.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:LONGitude<CH>
```

class LongitudeCls

Longitude commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:LONGitude<CH>
value: float = driver.source.bb.isdbt.eew.longitude.get(index = repcap.Index.
↪Default)
```

Sets the geographical longitude of the epicenter of the seismic event.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Longitude')

return
longitude: float Range: -180.0 DEG to 180.0 DEG

set(*longitude: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:LONGitude<CH>
driver.source.bb.isdbt.eew.longitude.set(longitude = 1.0, index = repcap.Index.
↪Default)
```

Sets the geographical longitude of the epicenter of the seismic event.

param longitude
float Range: -180.0 DEG to 180.0 DEG

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Longitude')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.eew.longitude.clone()
```

6.18.3.17.3.8 Occurence<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.isdbt.eew.occurence.repcap_index_get()
driver.source.bb.isdbt.eew.occurence.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:OCCurence<CH>
```

class OccurenceCls

Occurence commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:OCCurence<CH>
value: float = driver.source.bb.isdbt.eew.occurence.get(index = repcap.Index.
↪Default)
```

Sets the occurrence time of the seismic event.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Occurence')

return
occurence: float Range: 0 S to 1023 S

set(*occurence: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:OCCurence<CH>
driver.source.bb.isdbt.eew.occurence.set(occurence = 1.0, index = repcap.Index.
↪Default)
```

Sets the occurrence time of the seismic event.

param occurence
float Range: 0 S to 1023 S

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Occurence')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.eew.occurence.clone()
```

6.18.3.17.3.9 WarnId<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.isdbt.eew.warnId.repcap_index_get()
driver.source.bb.isdbt.eew.warnId.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:EEW:WARNid<CH>
```

class WarnIdCls

WarnId commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:WARNid<CH>
value: int = driver.source.bb.isdbt.eew.warnId.get(index = repcap.Index.Default)
```

Sets the individual identification number of the seismic motion warning.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'WarnId')

return
warning_id: integer Range: 0 to 511

set(*warning_id: int*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:EEW:WARNid<CH>
driver.source.bb.isdbt.eew.warnId.set(warning_id = 1, index = repcap.Index.
↪Default)
```

Sets the individual identification number of the seismic motion warning.

param warning_id
integer Range: 0 to 511

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'WarnId')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.eew.warnId.clone()
```

6.18.3.17.4 Fft

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:FFT:MODE
```

class FftCls

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

get_mode() → CodingIsdbtMode

```
# SCPI: [SOURce<HW>]:BB:ISDBt:FFT:MODE
value: enums.CodingIsdbtMode = driver.source.bb.isdbt.fft.get_mode()
```

Sets the ISDB-T mode.

return
isdbt_mode: M3_8K| M2_4K| M1_2K

set_mode(*isdbt_mode: CodingIsdbtMode*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:FFT:MODE
driver.source.bb.isdbt.fft.set_mode(isdbt_mode = enums.CodingIsdbtMode.M1_2K)
```

Sets the ISDB-T mode.

param isdbt_mode
M3_8K| M2_4K| M1_2K

6.18.3.17.5 Iip

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:IIP:PID
```

class IipCls

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

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:IIP:PID
value: int = driver.source.bb.isdbt.iip.get_pid()
```

Defines the for packets, that contain ISDB-T initialization packet (IIP) data.

return
iip_pid: integer Range: #H0000 to #H1FFF

set_pid(*iip_pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:IIP:PID
driver.source.bb.isdbt.iip.set_pid(iip_pid = 1)
```

Defines the for packets, that contain ISDB-T initialization packet (IIP) data.

param iip_pid
integer Range: #H0000 to #H1FFF

6.18.3.17.6 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:INPut:FORMat
[SOURce<HW>]:BB:ISDBt:INPut:TSCHannel
[SOURce<HW>]:BB:ISDBt:[INPut]:DATarate
[SOURce<HW>]:BB:ISDBt:INPut
```

class InputPyCls

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

get_data_rate() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[INPut]:DATarate
value: int = driver.source.bb.isdbt.inputPy.get_data_rate()


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
meas_drate: integer Range: 0 to 9999

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:ISDBt:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.isdbt.inputPy.get_format_py()
```

Sets the format of the input signal.

return
input_format: ASI| SMPTE

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:ISDBt:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.isdbt.inputPy.get_ts_channel()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

return
ts_channel: 1| 2| 3| 4

get_value() → CodingInputSignalInputB

```
# SCPI: [SOURce<HW>]:BB:ISDBt:INPut
value: enums.CodingInputSignalInputB = driver.source.bb.isdbt.inputPy.get_
↪value()
```

Sets the external input interface.

return
input_py: TS| ASIFront| ASIRear| SPIFront| SPIRear| IP

set_format_py(*input_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:INPut:FORMat
driver.source.bb.isdbt.inputPy.set_format_py(input_format = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param input_format
ASI| SMPTE

set_ts_channel(*ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:INPut:TSCHannel
driver.source.bb.isdbt.inputPy.set_ts_channel(ts_channel = enums.NumberA._1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param ts_channel
1| 2| 3| 4

set_value(*input_py: CodingInputSignalInputB*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:INPut
driver.source.bb.isdbt.inputPy.set_value(input_py = enums.
↪CodingInputSignalInputB.ASIFront)
```

Sets the external input interface.

param input_py
TS| ASIFront| ASIRear| SPIFront| SPIRear| IP

6.18.3.17.7 Payload

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:PAYLoad:A
```

class PayloadCls

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

get_a() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PAYLoad:A
value: enums.PayloadTestStuff = driver.source.bb.isdbt.payload.get_a()
```

Defines the payload area content of the packet.

return
payload: HFF| H00| PRBS

set_a(*payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PAYLoad:A
driver.source.bb.isdbt.payload.set_a(payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param payload
HFF| H00| PRBS

6.18.3.17.8 Prbs

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:PRBS:[SEQuence]
```

class PrbsCls

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

get_sequence() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PRBS:[SEQuence]
value: enums.SettingsPrbs = driver.source.bb.isdbt.prbs.get_sequence()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
prbs: P15_1| P23_1

set_sequence(*prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:PRBS:[SEQuence]
driver.source.bb.isdbt.prbs.set_sequence(prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param prbs
P15_1| P23_1

6.18.3.17.9 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:RATE:A
[SOURce<HW>]:BB:ISDBt:RATE:B
[SOURce<HW>]:BB:ISDBt:RATE:C
```

class RateCls

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

get_a() → CodingCoderate

```
# SCPI: [SOURce<HW>]:BB:ISDBt:RATE:A
value: enums.CodingCoderate = driver.source.bb.isdbt.rate.get_a()
```

Sets the code rate.

return
coderate_a: No help available

get_b() → CodingCoderate

```
# SCPI: [SOURce<HW>]:BB:ISDBt:RATE:B
value: enums.CodingCoderate = driver.source.bb.isdbt.rate.get_b()
```

Sets the code rate.

return
coderate_b: No help available

get_c() → CodingCoderate

```
# SCPI: [SOURce<HW>]:BB:ISDBt:RATE:C
value: enums.CodingCoderate = driver.source.bb.isdbt.rate.get_c()
```

Sets the code rate.

return
coderate_c: R7_8| R5_6| R3_4| R2_3| R1_2

set_a(*coderate_a: CodingCoderate*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:RATE:A
driver.source.bb.isdbt.rate.set_a(coderate_a = enums.CodingCoderate.R1_2)
```

Sets the code rate.

param coderate_a
R7_8| R5_6| R3_4| R2_3| R1_2

set_b(*coderate_b: CodingCoderate*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:RATE:B
driver.source.bb.isdbt.rate.set_b(coderate_b = enums.CodingCoderate.R1_2)
```

Sets the code rate.

param coderate_b
R7_8| R5_6| R3_4| R2_3| R1_2

set_c(*coderate_c: CodingCoderate*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:RATE:C
driver.source.bb.isdbt.rate.set_c(coderate_c = enums.CodingCoderate.R1_2)
```

Sets the code rate.

param coderate_c
R7_8| R5_6| R3_4| R2_3| R1_2

6.18.3.17.10 Segments

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:SEGMents:A
[SOURce<HW>]:BB:ISDBt:SEGMents:B
[SOURce<HW>]:BB:ISDBt:SEGMents:C
```

class SegmentsCls

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

get_a() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SEGMents:A
value: int = driver.source.bb.isdbt.segments.get_a()
```

Sets the number of segments for layers A, B and C.

return
segments_a: No help available

get_b() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SEGMents:B
value: int = driver.source.bb.isdbt.segments.get_b()
```

Sets the number of segments for layers A, B and C.

return
segments_b: No help available

get_c() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SEGMents:C
value: int = driver.source.bb.isdbt.segments.get_c()
```

Sets the number of segments for layers A, B and C.

return
segments_c: integer Range: 0 to 11

set_a(*segments_a: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SEGMents:A
driver.source.bb.isdbt.segments.set_a(segments_a = 1)
```

Sets the number of segments for layers A, B and C.

param segments_a
integer Range: 0 to 11

set_b(*segments_b: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SEGMents:B
driver.source.bb.isdbt.segments.set_b(segments_b = 1)
```

Sets the number of segments for layers A, B and C.

param segments_b
integer Range: 0 to 11

set_c(*segments_c: int*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SEGMents:C
driver.source.bb.isdbt.segments.set_c(segments_c = 1)
```

Sets the number of segments for layers A, B and C.

param segments_c
integer Range: 0 to 11

6.18.3.17.11 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:SETTing:CATalog
[SOURce<HW>]:BB:ISDBt:SETTing:DELete
[SOURce<HW>]:BB:ISDBt:SETTing:LOAD
[SOURce<HW>]:BB:ISDBt:SETTing:STORe
```

class SettingCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SETTing:CATalog
value: List[str] = driver.source.bb.isdbt.setting.get_catalog()
```

No command help available

return
isdbt_cat_name: No help available

get_delete() → str

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SETTing:DELete
value: str = driver.source.bb.isdbt.setting.get_delete()
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.isdbt. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
delete: 'filename' Filename or complete file path; file extension can be omitted

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SETTing:LOAD
value: str = driver.source.bb.isdbt.setting.get_load()
```

Accesses the 'Save/Recall' dialog, that is the standard instrument function for saving and recalling the complete dialog-related settings in a file. The provided navigation possibilities in the dialog are self-explanatory. The settings are saved in a file with predefined extension. You can define the filename and the directory, in that you want to save the file. .

return
isdbt_recall: string

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SETTing:STORe
value: str = driver.source.bb.isdbt.setting.get_store()
```

Accesses the 'Save/Recall' dialog, that is the standard instrument function for saving and recalling the complete dialog-related settings in a file. The provided navigation possibilities in the dialog are self-explanatory. The settings are saved in a file with predefined extension. You can define the filename and the directory, in that you want to save the file. .

return
isdbt_save: string

set_delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SETTing:DELete
driver.source.bb.isdbt.setting.set_delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.isdbt. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

set_load(*isdbt_recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SETTing:LOAD
driver.source.bb.isdbt.setting.set_load(isdbt_recall = 'abc')
```

Accesses the 'Save/Recall' dialog, that is the standard instrument function for saving and recalling the complete dialog-related settings in a file. The provided navigation possibilities in the dialog are self-explanatory.

The settings are saved in a file with predefined extension. You can define the filename and the directory, in that you want to save the file. .

param isdbt_recall
string

set_store(*isdbt_save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SETTing:STORe
driver.source.bb.isdbt.setting.set_store(isdbt_save = 'abc')
```

Accesses the 'Save/Recall' dialog, that is the standard instrument function for saving and recalling the complete dialog-related settings in a file. The provided navigation possibilities in the dialog are self-explanatory. The settings are saved in a file with predefined extension. You can define the filename and the directory, in that you want to save the file. .

param isdbt_save
string

6.18.3.17.12 Source

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:SOURce:A
[SOURce<HW>]:BB:ISDBt:SOURce:B
[SOURce<HW>]:BB:ISDBt:SOURce:C
```

class SourceCls

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

get_a() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SOURce:A
value: enums.CodingInputSignalSource = driver.source.bb.isdbt.source.get_a()
```

Sets the modulation source for layer A, B or C.

return
source_a: No help available

get_b() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SOURce:B
value: enums.CodingInputSignalSource = driver.source.bb.isdbt.source.get_b()
```

Sets the modulation source for layer A, B or C.

return
source_b: No help available

get_c() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SOURce:C
value: enums.CodingInputSignalSource = driver.source.bb.isdbt.source.get_c()
```

Sets the modulation source for layer A, B or C.

return
source_c: TESTsignal| TSPLayer| EXTernal

set_a(*source_a: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SOURce:A
driver.source.bb.isdbt.source.set_a(source_a = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for layer A, B or C.

param source_a
TESTsignal| TSPLayer| EXTernal

set_b(*source_b: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SOURce:B
driver.source.bb.isdbt.source.set_b(source_b = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for layer A, B or C.

param source_b
TESTsignal| TSPLayer| EXTernal

set_c(*source_c: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:SOURce:C
driver.source.bb.isdbt.source.set_c(source_c = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for layer A, B or C.

param source_c
TESTsignal| TSPLayer| EXTernal

6.18.3.17.13 Special

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:[SPECial]:ACData2
[SOURce<HW>]:BB:ISDBt:[SPECial]:REEDsolomon
[SOURce<HW>]:BB:ISDBt:[SPECial]:TXParam
```

class SpecialCls

Special commands group definition. 6 total commands, 3 Subgroups, 3 group commands

get_ac_data_2() → SpecialAcData

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:ACData2
value: enums.SpecialAcData = driver.source.bb.isdbt.special.get_ac_data_2()
```

Sets the carrier modulation.

return
ac_data_2: ALL1| PRBS ALL1 Sets all carriers to 1. PRBS Sets PRBS modulated carriers. You can set the PRBS length via [:SOURcehw]:BB:ISDBt:PRBS[:SEQuence].

get_reed_solomon() → bool

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:REEDsolomon
value: bool = driver.source.bb.isdbt.special.get_reed_solomon()
```

Enables/disables the Reed-Solomon encoder.

return
reed_solomon: 1| ON| 0| OFF

get_tx_param() → IsdbtSpecialTxParam

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:TXParam
value: enums.IsdbtSpecialTxParam = driver.source.bb.isdbt.special.get_tx_param()
```

Defines the static setting of the transmission parameter switching indicator.

return
tx_param_sw_ind: N1| N2| N11| N12| N13| N14| N15| NORMal| N2| N4| N5| N6| N7| N8| N9| N10

set_ac_data_2(*ac_data_2: SpecialAcData*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:ACData2
driver.source.bb.isdbt.special.set_ac_data_2(ac_data_2 = enums.SpecialAcData.
↪ALL1)
```

Sets the carrier modulation.

param ac_data_2
ALL1| PRBS ALL1 Sets all carriers to 1. PRBS Sets PRBS modulated carriers. You can set the PRBS length via [:SOURcehw]:BB:ISDBt:PRBS[:SEQuence].

set_reed_solomon(*reed_solomon: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:REEDsolomon
driver.source.bb.isdbt.special.set_reed_solomon(reed_solomon = False)
```

Enables/disables the Reed-Solomon encoder.

param reed_solomon
1| ON| 0| OFF

set_tx_param(*tx_param_sw_ind: IsdbtSpecialTxParam*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:TXParam
driver.source.bb.isdbt.special.set_tx_param(tx_param_sw_ind = enums.
↪IsdbtSpecialTxParam.N1)
```

Defines the static setting of the transmission parameter switching indicator.

param tx_param_sw_ind
N1| N2| N11| N12| N13| N14| N15| NORMal| N2| N4| N5| N6| N7| N8| N9| N10

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.special.clone()
```

Subgroups

6.18.3.17.13.1 Alert

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:[SPECial]:ALERt:[BROadcast]
```

class AlertCls

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

get_broadcast() → bool

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:ALERt:[BROadcast]
value: bool = driver.source.bb.isdbt.special.alert.get_broadcast()
```

Enables or disables the alert broadcasting flag in the data.

return
alert_bc_flag: 1| ON| 0| OFF

set_broadcast(*alert_bc_flag: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:ALERt:[BROadcast]
driver.source.bb.isdbt.special.alert.set_broadcast(alert_bc_flag = False)
```

Enables or disables the alert broadcasting flag in the data.

param alert_bc_flag
1| ON| 0| OFF

6.18.3.17.13.2 Settings

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:[SPECial]:SETTings:[STATe]
```

class SettingsCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:SETTings:[STATe]
value: bool = driver.source.bb.isdbt.special.settings.get_state()
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

return
settings: 1| ON| 0| OFF

set_state(*settings: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:SETTings:[STATe]
driver.source.bb.isdbt.special.settings.set_state(settings = False)
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

param settings
1| ON| 0| OFF

6.18.3.17.13.3 Tmcc

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:[SPECial]:TMCC:NEXT
```

class TmccCls

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

get_next() → IsdbtSpecialTmcc

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:TMCC:NEXT
value: enums.IsdbtSpecialTmcc = driver.source.bb.isdbt.special.tmcc.get_next()
```

Sets the next information bits.

return
mtcc_next: UNUSed| CURRent

set_next(*mtcc_next: IsdbtSpecialTmcc*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:[SPECial]:TMCC:NEXT
driver.source.bb.isdbt.special.tmcc.set_next(mtcc_next = enums.IsdbtSpecialTmcc.
↪CURRent)
```

Sets the next information bits.

param mtcc_next
UNUSed| CURRent

6.18.3.17.14 TestSignal

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:TESTsignal:A
[SOURce<HW>]:BB:ISDBt:TESTsignal:B
[SOURce<HW>]:BB:ISDBt:TESTsignal:C
```

class TestSignalCls

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

get_a() → InputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TESTsignal:A
value: enums.InputSignalTestSignal = driver.source.bb.isdbt.testSignal.get_a()
```

Defines the test signal data.

return
test_signal_a: No help available

get_b() → InputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TESTsignal:B
value: enums.InputSignalTestSignal = driver.source.bb.isdbt.testSignal.get_b()
```

Defines the test signal data.

return
test_signal_b: No help available

get_c() → InputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TESTsignal:C
value: enums.InputSignalTestSignal = driver.source.bb.isdbt.testSignal.get_c()
```

Defines the test signal data.

return
test_signal_c: PAFC| PBEC| TTSP

set_a(*test_signal_a: InputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TESTsignal:A
driver.source.bb.isdbt.testSignal.set_a(test_signal_a = enums.
↪InputSignalTestSignal.PAFC)
```

Defines the test signal data.

param test_signal_a
PAFC| PBEC| TTSP

set_b(*test_signal_b: InputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TESTsignal:B
driver.source.bb.isdbt.testSignal.set_b(test_signal_b = enums.
↪InputSignalTestSignal.PAFC)
```

Defines the test signal data.

param test_signal_b
PAFC| PBEC| TTSP

set_c(*test_signal_c: InputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TESTsignal:C
driver.source.bb.isdbt.testSignal.set_c(test_signal_c = enums.
↪InputSignalTestSignal.PAFC)
```

Defines the test signal data.

param test_signal_c
PAFC| PBEC| TTSP

6.18.3.17.15 Time

class TimeCls

Time 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.source.bb.isdbt.time.clone()
```

Subgroups

6.18.3.17.15.1 Interleaving

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:A
[SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:B
[SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:C
```

class InterleavingCls

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

get_a() → CodingTimeInterleaving

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:A
value: enums.CodingTimeInterleaving = driver.source.bb.isdbt.time.interleaving.
↪get_a()
```

Sets the time interleaving depth of each layer separately.

return
time_int_a: No help available

get_b() → CodingTimeInterleaving

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:B
value: enums.CodingTimeInterleaving = driver.source.bb.isdbt.time.interleaving.
↪get_b()
```

Sets the time interleaving depth of each layer separately.

return
time_int_b: No help available

get_c() → CodingTimeInterleaving

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:C
value: enums.CodingTimeInterleaving = driver.source.bb.isdbt.time.interleaving.
↪get_c()
```

Sets the time interleaving depth of each layer separately.

return
time_int_c: 0| 1| 16| 2| 32| 4| 8

set_a(*time_int_a: CodingTimeInterleaving*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:A
driver.source.bb.isdbt.time.interleaving.set_a(time_int_a = enums.
↪CodingTimeInterleaving._0)
```

Sets the time interleaving depth of each layer separately.

param time_int_a
0| 1| 16| 2| 32| 4| 8

set_b(*time_int_b: CodingTimeInterleaving*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:B
driver.source.bb.isdbt.time.interleaving.set_b(time_int_b = enums.
↪CodingTimeInterleaving._0)
```

Sets the time interleaving depth of each layer separately.

param time_int_b
0| 1| 16| 2| 32| 4| 8

set_c(*time_int_c: CodingTimeInterleaving*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TIME:[INTerleaving]:C
driver.source.bb.isdbt.time.interleaving.set_c(time_int_c = enums.
↪CodingTimeInterleaving._0)
```

Sets the time interleaving depth of each layer separately.

param time_int_c
0| 1| 16| 2| 32| 4| 8

6.18.3.17.16 TsPackets

SCPI Command :

```
[SOURce<HW>]:BB:ISDBt:TSPackets:A
```

class TsPacketsCls

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

get_a() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TSPackets:A
value: enums.SettingsTestTsPacket = driver.source.bb.isdbt.tsPackets.get_a()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
test_ts_packet: S187| H184

set_a(*test_ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:ISDBt:TSPackets:A
driver.source.bb.isdbt.tsPackets.set_a(test_ts_packet = enums.
↪SettingsTestTsPacket.H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param test_ts_packet
S187| H184

6.18.3.17.17 Useful

class UsefulCls

Useful commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.useful.clone()
```

Subgroups

6.18.3.17.17.1 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:A
[SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:B
[SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:C
```

class RateCls

Rate commands group definition. 6 total commands, 1 Subgroups, 3 group commands

get_a() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:A
value: int = driver.source.bb.isdbt.useful.rate.get_a()
```

Displays the data rate measured in the specific layer.

return
meas_use_drate_a: integer Range: 0 to 9999

get_b() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:B
value: int = driver.source.bb.isdbt.useful.rate.get_b()
```

Displays the data rate measured in the specific layer.

return
meas_use_drate_b: integer Range: 0 to 9999

get_c() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:C
value: int = driver.source.bb.isdbt.useful.rate.get_c()
```

Displays the data rate measured in the specific layer.

return
meas_use_drata_c: integer Range: 0 to 9999

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.isdbt.useful.rate.clone()
```

Subgroups

6.18.3.17.17.2 Max

SCPI Commands :

```
[SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:MAX:A
[SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:MAX:B
[SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:MAX:C
```

class MaxCls

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

get_a() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:MAX:A
value: int = driver.source.bb.isdbt.useful.rate.max.get_a()
```

Displays the maximum useful data rate in the specific layer.

return
max_use_drate_a: integer Range: 0 to 999

get_b() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:MAX:B
value: int = driver.source.bb.isdbt.useful.rate.max.get_b()
```

Displays the maximum useful data rate in the specific layer.

return
max_use_drate_b: integer Range: 0 to 999

get_c() → int

```
# SCPI: [SOURce<HW>]:BB:ISDBt:USEFul:[RATE]:MAX:C
value: int = driver.source.bb.isdbt.useful.rate.max.get_c()
```

Displays the maximum useful data rate in the specific layer.

return
max_use_drate_c: integer Range: 0 to 999

6.18.3.18 J83B

SCPI Commands :

```
[SOURce<HW>]:BB:J83B:CONStel
[SOURce<HW>]:BB:J83B:PACKetlength
[SOURce<HW>]:BB:J83B:PAYLoad
[SOURce<HW>]:BB:J83B:PID
[SOURce<HW>]:BB:J83B:PIDTestpack
[SOURce<HW>]:BB:J83B:PRBS
[SOURce<HW>]:BB:J83B:PRESet
[SOURce<HW>]:BB:J83B:ROLLoff
[SOURce<HW>]:BB:J83B:SOURce
[SOURce<HW>]:BB:J83B:STATe
[SOURce<HW>]:BB:J83B:STUFfing
[SOURce<HW>]:BB:J83B:SYMBols
[SOURce<HW>]:BB:J83B:TESTsignal
[SOURce<HW>]:BB:J83B:TSPacket
```

class J83BCls

J83B commands group definition. 27 total commands, 5 Subgroups, 14 group commands

get_constel() → J83BcodingJ83BcodingConstel

```
# SCPI: [SOURce<HW>]:BB:J83B:CONStel
value: enums.J83BcodingJ83BcodingConstel = driver.source.bb.j83B.get_constel()
```

Sets the constellation for modulation schemes.

return
constel: J64| J256| J1024 J64 Modulation setting QAM64. J256 Modulation setting QAM256. J1024 Modulation setting QAM1024.

get_packet_length() → CodingInputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:J83B:PACKetlength
value: enums.CodingInputSignalPacketLength = driver.source.bb.j83B.get_packet_
↪length()
```

Queries the packet length of the external transport stream in bytes.

return
inp_sig_plength: P188| INValid P188 188 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:J83B:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.j83B.get_payload()
```

Sets the payload field of the transport stream packet content.

return
set_payload: PRBS| H00| HFF PRBS PRBS data in accordance with H00 Exclusively 00 (hex) data HFF Exclusively FF (hex) data

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:J83B:PID
value: int = driver.source.bb.j83B.get_pid()
```

Sets the packet identifier (PID) .

return
set_pid: integer Range: 0 to 8191

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:J83B:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.j83B.get_pid_test_pack()
```

Sets a fixed or variable packet identifier (PID) . If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
set_pid_testpack: NULL| VARiable

get_prbs() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:J83B:PRBS
value: enums.SettingsPrbs = driver.source.bb.j83B.get_prbs()
```

Sets the length of the PRBS sequence.

return
set_prbs: P23_1| P15_1 P23_1 PRBS 23 sequence as specified by . P15_1 PRBS 15 sequence as specified by .

get_rolloff() → J83BcodingJ83BcodingRolloff

```
# SCPI: [SOURce<HW>]:BB:J83B:ROLLoff
value: enums.J83BcodingJ83BcodingRolloff = driver.source.bb.j83B.get_rolloff()
```

Queries the roll-off factor.

return
rolloff: 0.12| 0.18

get_source() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:J83B:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.j83B.get_source()
```

Sets the modulation source for the input signal.

return
inp_sig_source: EXTernal| TSPLayer| TESTsignal

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:J83B:STATe
value: bool = driver.source.bb.j83B.get_state()
```

Enables/disables the DVB-S standard.

return
state: 1| ON| 0| OFF

get_stuffing() → StateOn

```
# SCPI: [SOURce<HW>]:BB:J83B:STUFfing
value: enums.StateOn = driver.source.bb.j83B.get_stuffing()
```

Queries the stuffing state that is active.

return
inp_sig_stuffing: 1| ON

get_symbols() → int

```
# SCPI: [SOURce<HW>]:BB:J83B:SYMBols
value: int = driver.source.bb.j83B.get_symbols()
```

Sets the symbol rate.

return
symbol_rate: integer Range: 4.5512469E+06 to 5.8965907E+06

get_test_signal() → J83BcodingJ83BinputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:J83B:TESTsignal
value: enums.J83BcodingJ83BinputSignalTestSignal = driver.source.bb.j83B.get_
↪test_signal()
```

Defines the test signal data.

return
inp_sig_test_sig: TTSP| PBEM| PBTR TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBDE Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the convolutional encoder. PRBS data conforms with specification. PBEM PRBS after convolutional encoder Pure pseudo-random bit sequence (PRBS)

data used as modulation data with no packet structure. The sequence is inserted after the convolutional encoder.

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:J83B:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.j83B.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
set_ts_packet: H184| S187 H184 Head/184 Payload S187 Sync/187 Payload

preset() → None

```
# SCPI: [SOURce<HW>]:BB:J83B:PRESet
driver.source.bb.j83B.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>:BB:J83B:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:J83B:PRESet
driver.source.bb.j83B.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>:BB:J83B:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_constel(*constel: J83BcodingJ83BcodingConstel*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:CONStel
driver.source.bb.j83B.set_constel(constel = enums.J83BcodingJ83BcodingConstel.
↪J1024)
```

Sets the constellation for modulation schemes.

param constel
J64| J256| J1024 J64 Modulation setting QAM64. J256 Modulation setting QAM256. J1024 Modulation setting QAM1024.

set_payload(*set_payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:PAYLoad
driver.source.bb.j83B.set_payload(set_payload = enums.PayloadTestStuff.H00)
```

Sets the payload field of the transport stream packet content.

param set_payload
PRBS| H00| HFF PRBS PRBS data in accordance with H00 Exclusively 00 (hex) data HFF Exclusively FF (hex) data

set_pid(*set_pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:PID
driver.source.bb.j83B.set_pid(set_pid = 1)
```

Sets the packet identifier (PID) .

param set_pid
integer Range: 0 to 8191

set_pid_test_pack(*set_pid_testpack: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:PIDTestpack
driver.source.bb.j83B.set_pid_test_pack(set_pid_testpack = enums.PidTestPacket.
↪NULL)
```

Sets a fixed or variable packet identifier (PID) . If a header is present in the test packet ('Test TS Packet > Head/184 Payload') , you can specify a fixed or variable packet identifier (PID) .

param set_pid_testpack
NULL| VARiable

set_prbs(*set_prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:PRBS
driver.source.bb.j83B.set_prbs(set_prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence.

param set_prbs
P23_1| P15_1 P23_1 PRBS 23 sequence as specified by . P15_1 PRBS 15 sequence as specified by .

set_source(*inp_sig_source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:SOURce
driver.source.bb.j83B.set_source(inp_sig_source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param inp_sig_source
EXTernal| TSPLayer| TESTsignal

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:STATe
driver.source.bb.j83B.set_state(state = False)
```

Enables/disables the DVB-S standard.

param state
1| ON| 0| OFF

set_symbols(*symbol_rate: int*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:SYMBols
driver.source.bb.j83B.set_symbols(symbol_rate = 1)
```

Sets the symbol rate.

param symbol_rate
integer Range: 4.5512469E+06 to 5.8965907E+06

set_test_signal(*inp_sig_test_sig: J83BcodingJ83BinputSignalTestSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:TESTsignal
driver.source.bb.j83B.set_test_signal(inp_sig_test_sig = enums.
↪J83BcodingJ83BinputSignalTestSignal.PBEM)
```

Defines the test signal data.

param inp_sig_test_sig
TTSP| PBEM| PBTR TTSP Test TS packet with standardized packet data used as modulation data in the transport stream. PBDE Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted before the convolutional encoder. PRBS data conforms with specification. PBEM PRBS after convolutional encoder Pure pseudo-random bit sequence (PRBS) data used as modulation data with no packet structure. The sequence is inserted after the convolutional encoder.

set_ts_packet(*set_ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:TSPacket
driver.source.bb.j83B.set_ts_packet(set_ts_packet = enums.SettingsTestTsPacket.
↪H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param set_ts_packet
H184| S187 H184 Head/184 Payload S187 Sync/187 Payload

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.j83B.clone()
```

Subgroups

6.18.3.18.1 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:J83B:INPut:FORMat
[SOURce<HW>]:BB:J83B:INPut:TSCHannel
[SOURce<HW>]:BB:J83B:[INPut]:DATarate
[SOURce<HW>]:BB:J83B:INPut
```

class InputPyCls

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

get_data_rate() → float

```
# SCPI: [SOURce<HW>]:BB:J83B:[INPut]:DATarate
value: float = driver.source.bb.j83B.inputPy.get_data_rate()
```

Queries the measured value of the data rate of one of the following: External transport stream including null packets input at 'User 1' connector External transport stream including null packets input at 'IP Data/LAN' connector (TSoverIP) The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

return
inp_sig_datarate: float Range: 0 to 999999999

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:J83B:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.j83B.inputPy.get_format_py()
```

Sets the format of the input signal.

return
inp_sig_format: ASI| SMPTE

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:J83B:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.j83B.inputPy.get_ts_channel()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

return
inp_sig_ts_channel: 1| 2| 3| 4

get_value() → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:J83B:INPut
value: enums.CodingInputSignalInputA = driver.source.bb.j83B.inputPy.get_value()
```

Sets the external input interface.

return
inp_sig_input: TS| IP TS Input for serial transport stream data. The signal is input at the 'User 1/2' connectors. IP Supported for high priority path (HP) only, i.e. setting requires non-hierarchical coding. Input for IP transport stream data. The signal is input at the 'IP Data' connector.

set_format_py(*inp_sig_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:INPut:FORMat
driver.source.bb.j83B.inputPy.set_format_py(inp_sig_format = enums.
↪CodingInputFormat.ASI)
```

Sets the format of the input signal.

param inp_sig_format
ASI| SMPTE

set_ts_channel(*inp_sig_ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:INPut:TSCHannel
driver.source.bb.j83B.inputPy.set_ts_channel(inp_sig_ts_channel = enums.NumberA.
↪_1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param inp_sig_ts_channel
1| 2| 3| 4

set_value(*inp_sig_input: CodingInputSignalInputA*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:INPut
driver.source.bb.j83B.inputPy.set_value(inp_sig_input = enums.
↪CodingInputSignalInputA.ASI1)
```

Sets the external input interface.

param inp_sig_input
TS| IP TS Input for serial transport stream data. The signal is input at the 'User 1/2' connectors. IP Supported for high priority path (HP) only, i.e. setting requires non-hierarchical coding. Input for IP transport stream data. The signal is input at the 'IP Data' connector.

6.18.3.18.2 Interleaver

SCPI Command :

```
[SOURce<HW>]:BB:J83B:INTerleaver:MODE
```

class InterleaverCls

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

get_mode() → int

```
# SCPI: [SOURce<HW>]:BB:J83B:INTerleaver:MODE
value: int = driver.source.bb.j83B.interleaver.get_mode()
```

Sets the interleaver mode.

return
interleaver_mode: No help available

set_mode(*interleaver_mode: int*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:INTerleaver:MODE
driver.source.bb.j83B.interleaver.set_mode(interleaver_mode = 1)
```

Sets the interleaver mode.

param interleaver_mode
No help available

6.18.3.18.3 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:J83B:SETTing:CATalog
[SOURce<HW>]:BB:J83B:SETTing:DELete
[SOURce<HW>]:BB:J83B:SETTing:LOAD
[SOURce<HW>]:BB:J83B:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:SETTing:DELete
driver.source.bb.j83B.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.J83B. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:J83B:SETTing:CATalog
value: List[str] = driver.source.bb.j83B.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.J83B. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
catalog: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:J83B:SETTing:LOAD
value: str = driver.source.bb.j83B.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.J83B. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: 'filename' Filename or complete file path; file extension can be omitted.

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:J83B:SETTing:STORe
value: str = driver.source.bb.j83B.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.J83B) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: 'filename' Filename or complete file path

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:SETTing:LOAD
driver.source.bb.j83B.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.J83B. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
'filename' Filename or complete file path; file extension can be omitted.

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:SETTing:STORe
driver.source.bb.j83B.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.J83B) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
'filename' Filename or complete file path

6.18.3.18.4 Special

SCPI Command :

```
[SOURce<HW>]:BB:J83B:[SPECial]:REEDsolomon
```

class SpecialCls

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

get_reed_solomon() → bool

```
# SCPI: [SOURce<HW>]:BB:J83B:[SPECial]:REEDsolomon
value: bool = driver.source.bb.j83B.special.get_reed_solomon()
```

Enables/disables the Reed-Solomon encoder.

return
reed_solomon: 1| ON| 0| OFF

set_reed_solomon(*reed_solomon: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:[SPECial]:REEDsolomon
driver.source.bb.j83B.special.set_reed_solomon(reed_solomon = False)
```

Enables/disables the Reed-Solomon encoder.

param reed_solomon
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.j83B.special.clone()
```

Subgroups

6.18.3.18.4.1 Setting

SCPI Command :

```
[SOURce<HW>]:BB:J83B:[SPECial]:SETTing:[STATe]
```

class SettingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:J83B:[SPECial]:SETTing:[STATe]
value: bool = driver.source.bb.j83B.special.setting.get_state()
```

Enables/disables special settings.

return
special_state: 1| ON| 0| OFF

set_state(*special_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:J83B:[SPECial]:SETTing:[STATe]
driver.source.bb.j83B.special.setting.set_state(special_state = False)
```

Enables/disables special settings.

param special_state
1| ON| 0| OFF

6.18.3.18.5 Useful

class UsefulCls

Useful 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.source.bb.j83B.useful.clone()
```

Subgroups

6.18.3.18.5.1 Rate

SCPI Commands :

```
[SOURce<HW>]:BB:J83B:USEFul:[RATE]:MAX
[SOURce<HW>]:BB:J83B:USEFul:[RATE]
```

class RateCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:BB:J83B:USEFul:[RATE]:MAX
value: float = driver.source.bb.j83B.useful.rate.get_max()
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator If 'Stuffing > On', the value indicates the maximum useful data rate, that is allowed in the transport stream. If 'Stuffing > Off', the value indicates the transport stream input data rate that is required for the transport stream.

return
inp_sig_max_rate: float Range: 0 to 999999999

get_value() → float

```
# SCPI: [SOURce<HW>]:BB:J83B:USEFul:[RATE]
value: float = driver.source.bb.j83B.useful.rate.get_value()
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

return
inp_sig_usefull: float Range: 0 to 999999999

6.18.3.19 Lora

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:BWIDth
[SOURce<HW>]:BB:LORA:IINTerval
[SOURce<HW>]:BB:LORA:OSAMpling
[SOURce<HW>]:BB:LORA:PRESet
[SOURce<HW>]:BB:LORA:SLENgth
[SOURce<HW>]:BB:LORA:STATe
```

class LoraCls

Lora commands group definition. 61 total commands, 7 Subgroups, 6 group commands

get_bandwidth() → LoRaBw

```
# SCPI: [SOURce<HW>]:BB:LORA:BWIDth
value: enums.LoRaBw = driver.source.bb.lora.get_bandwidth()
```

No command help available

return
bw: No help available

get_iinterval() → float

```
# SCPI: [SOURce<HW>]:BB:LORA:IINTerval
value: float = driver.source.bb.lora.get_iinterval()
```

No command help available

return
iinterval: No help available

get_osampling() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:OSAMpling
value: int = driver.source.bb.lora.get_osampling()
```

No command help available

return
osampling: No help available

get_slength() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:SLENgth
value: int = driver.source.bb.lora.get_slength()
```

No command help available

return
slength: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:STATe
value: bool = driver.source.bb.lora.get_state()
```

No command help available

return
state: No help available

preset() → None

```
# SCPI: [SOURce<HW>]:BB:LORA:PRESet
driver.source.bb.lora.preset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:LORA:PRESet
driver.source.bb.lora.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_bandwidth(*bw: LoRaBw*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:BWIDth
driver.source.bb.lora.set_bandwidth(bw = enums.LoRaBw.BW10)
```

No command help available

param bw
No help available

set_iinterval(*iinterval: float*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IINTerval
driver.source.bb.lora.set_iinterval(iinterval = 1.0)
```

No command help available

param iinterval
No help available

set_osampling(*osampling: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:OSAMpling
driver.source.bb.lora.set_osampling(osampling = 1)
```

No command help available

param osampling
No help available

set_slength(*slength: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:SLENgth
driver.source.bb.lora.set_slength(slength = 1)
```

No command help available

param slength
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:STATe
driver.source.bb.lora.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.lora.clone()
```

Subgroups

6.18.3.19.1 Clock

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:CLOCk:MODE
[SOURce<HW>]:BB:LORA:CLOCk:MULTiplier
[SOURce<HW>]:BB:LORA:CLOCk:SOURce
```

class ClockCls

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

get_mode() → ClockModeUnits

```
# SCPI: [SOURce<HW>]:BB:LORA:CLOCk:MODE
value: enums.ClockModeUnits = driver.source.bb.lora.clock.get_mode()
```

No command help available

return
mode: No help available

get_multiplier() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:CLOCk:MULTiplier
value: int = driver.source.bb.lora.clock.get_multiplier()
```

No command help available

return
multiplier: No help available

get_source() → ClockSourceB

```
# SCPI: [SOURce<HW>]:BB:LORA:CLOCk:SOURce
value: enums.ClockSourceB = driver.source.bb.lora.clock.get_source()
```

No command help available

return
source: No help available

set_mode(*mode: ClockModeUnits*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:CLOCk:MODE
driver.source.bb.lora.clock.set_mode(mode = enums.ClockModeUnits.MSAMple)
```

No command help available

param mode
No help available

`set_multiplier`(*multiplier: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:CLOCk:MULTiplier
driver.source.bb.lora.clock.set_multiplier(multiplier = 1)
```

No command help available

param multiplier
No help available

`set_source`(*source: ClockSourceB*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:CLOCk:SOURce
driver.source.bb.lora.clock.set_source(source = enums.ClockSourceB.AINTernal)
```

No command help available

param source
No help available

6.18.3.19.2 Fconfiguration

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:CRATe
[SOURce<HW>]:BB:LORA:FCONfiguration:DLENgth
[SOURce<HW>]:BB:LORA:FCONfiguration:SFACtor
[SOURce<HW>]:BB:LORA:FCONfiguration:SMODe
[SOURce<HW>]:BB:LORA:FCONfiguration:UPLength
```

`class FconfigurationCls`

Fconfiguration commands group definition. 16 total commands, 9 Subgroups, 5 group commands

`get_crate`() → LoRaCodRate

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:CRATe
value: enums.LoRaCodRate = driver.source.bb.lora.fconfiguration.get_crate()
```

No command help available

return
crate: No help available

`get_dlength`() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DLENgth
value: int = driver.source.bb.lora.fconfiguration.get_dlength()
```

No command help available

return
dlength: No help available

get_sfactor() → LoRaSf

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:SFACtor
value: enums.LoRaSf = driver.source.bb.lora.fconfiguration.get_sfactor()
```

No command help available

return
sf: No help available

get_smode() → LoRaSyncMode

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:SMODe
value: enums.LoRaSyncMode = driver.source.bb.lora.fconfiguration.get_smode()
```

No command help available

return
smode: No help available

get_up_length() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:UPLength
value: int = driver.source.bb.lora.fconfiguration.get_up_length()
```

No command help available

return
plength: No help available

set_crate(*crate: LoRaCodRate*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:CRATe
driver.source.bb.lora.fconfiguration.set_crate(crate = enums.LoRaCodRate.CR0)
```

No command help available

param crate
No help available

set_dlength(*dlength: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DLENgth
driver.source.bb.lora.fconfiguration.set_dlength(dlength = 1)
```

No command help available

param dlength
No help available

set_sfactor(*sf: LoRaSf*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:SFACtor
driver.source.bb.lora.fconfiguration.set_sfactor(sf = enums.LoRaSf.SF10)
```

No command help available

param sf
No help available

set_smode(*smode: LoRaSyncMode*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:SMODe
driver.source.bb.lora.fconfiguration.set_smode(smode = enums.LoRaSyncMode.
↪PRIVate)
```

No command help available

param smode
No help available

set_up_length(*plength: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:UPLength
driver.source.bb.lora.fconfiguration.set_up_length(plength = 1)
```

No command help available

param plength
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.lora.fconfiguration.clone()
```

Subgroups

6.18.3.19.2.1 Bmode

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:BMODe:STATe
```

class BmodeCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:BMODe:STATe
value: bool = driver.source.bb.lora.fconfiguration.bmode.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:BMODe:STATe
driver.source.bb.lora.fconfiguration.bmode.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.2.2 Cmode

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:CMODe:STATe
```

class CmodeCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:CMODe:STATe
value: bool = driver.source.bb.lora.fconfiguration.cmode.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:CMODe:STATe
driver.source.bb.lora.fconfiguration.cmode.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.2.3 Data

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:DATA:DSELection
[SOURce<HW>]:BB:LORA:FCONfiguration:DATA
```

class DataCls

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

get_dselection() → str

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DATA:DSELection
value: str = driver.source.bb.lora.fconfiguration.data.get_dselection()
```

No command help available

return
dselection: No help available

get_value() → TdmaDataSource

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DATA
value: enums.TdmaDataSource = driver.source.bb.lora.fconfiguration.data.get_
↪value()
```

No command help available

return
data: No help available

set_dselection(*dselection: str*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DATA:DSELection
driver.source.bb.lora.fconfiguration.data.set_dselection(dselection = 'abc')
```

No command help available

param dselection
No help available

set_value(*data: TdmaDataSource*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DATA
driver.source.bb.lora.fconfiguration.data.set_value(data = enums.TdmaDataSource.
↪DLISt)
```

No command help available

param data
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.lora.fconfiguration.data.clone()
```

Subgroups

6.18.3.19.2.4 Dpattern

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:DATA:DPATtern
```

class DpatternCls

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

class DpatternStruct

Response structure. Fields:

- Dpattern: List[str]: No parameter help available
- Bitcount: int: No parameter help available

get() → DpatternStruct

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DATA:DPATtern
value: DpatternStruct = driver.source.bb.lora.fconfiguration.data.dpattern.get()
```

No command help available

return
structure: for return value, see the help for DpatternStruct structure arguments.

set(*dpattern: List[str]*, *bitcount: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:DATA:DPATtern
driver.source.bb.lora.fconfiguration.data.dpattern.set(dpattern = ['rawAbc1',
↪'rawAbc2', 'rawAbc3'], bitcount = 1)
```

No command help available

param dpattern
No help available

param bitcount
No help available

6.18.3.19.2.5 Eactive

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:EACTive:STATe
```

class EactiveCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:EACTive:STATe
value: bool = driver.source.bb.lora.fconfiguration.eactive.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:EACTive:STATe
driver.source.bb.lora.fconfiguration.eactive.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.2.6 Hactive

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:HACTive:STATe
```

class HactiveCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:HACTive:STATe
value: bool = driver.source.bb.lora.fconfiguration.hactive.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:HACTive:STATe
driver.source.bb.lora.fconfiguration.hactive.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.2.7 Iactive

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:IACTive:STATe
```

class IactiveCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:IACTive:STATe
value: bool = driver.source.bb.lora.fconfiguration.iactive.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:IACTive:STATe
driver.source.bb.lora.fconfiguration.iactive.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.2.8 Pcrc

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:PCRC:STATe
```

class PcrcCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:PCRC:STATe
value: bool = driver.source.bb.lora.fconfiguration.pcrc.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:PCRC:STATe
driver.source.bb.lora.fconfiguration.pcrc.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.2.9 PrcMode

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:PRCMode:STATe
```

class PrcModeCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:PRCMode:STATe
value: bool = driver.source.bb.lora.fconfiguration.prcMode.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:PRCMode:STATe
driver.source.bb.lora.fconfiguration.prcMode.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.2.10 Rbit

SCPI Command :

```
[SOURce<HW>]:BB:LORA:FCONfiguration:RBIT:STATe
```

class RbitCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:RBIT:STATe
value: bool = driver.source.bb.lora.fconfiguration.rbit.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:FCONfiguration:RBIT:STATe
driver.source.bb.lora.fconfiguration.rbit.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.3 Impairments

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:IMPairments:FDDeviation
[SOURce<HW>]:BB:LORA:IMPairments:FDRate
[SOURce<HW>]:BB:LORA:IMPairments:FDTYpe
[SOURce<HW>]:BB:LORA:IMPairments:FOFFset
[SOURce<HW>]:BB:LORA:IMPairments:STATe
[SOURce<HW>]:BB:LORA:IMPairments:STERror
```

class ImpairmentsCls

Impairments commands group definition. 7 total commands, 1 Subgroups, 6 group commands

get_fd_deviation() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDDeviation
value: int = driver.source.bb.lora.impairments.get_fd_deviation()
```

No command help available

return
fd_deviation: No help available

get_fd_rate() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDRate
value: int = driver.source.bb.lora.impairments.get_fd_rate()
```

No command help available

return
fd_rate: No help available

get_fd_type() → LoRaFreqDfTp

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDTYpe
value: enums.LoRaFreqDfTp = driver.source.bb.lora.impairments.get_fd_type()
```

No command help available

return
fd_type: No help available

get_foffset() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FOFFset
value: int = driver.source.bb.lora.impairments.get_foffset()
```

No command help available

return
foffset: No help available

get_st_error() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:STERror
value: int = driver.source.bb.lora.impairments.get_st_error()
```

No command help available

return
st_error: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:STATe
value: bool = driver.source.bb.lora.impairments.get_state()
```

No command help available

return
state: No help available

set_fd_deviation(*fd_deviation: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDDeviation
driver.source.bb.lora.impairments.set_fd_deviation(fd_deviation = 1)
```

No command help available

param fd_deviation
No help available

set_fd_rate(*fd_rate: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDRate
driver.source.bb.lora.impairments.set_fd_rate(fd_rate = 1)
```

No command help available

param fd_rate
No help available

set_fd_type(*fd_type: LoRaFreqDfTp*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDTYpe
driver.source.bb.lora.impairments.set_fd_type(fd_type = enums.LoRaFreqDfTp.
↪LINear)
```

No command help available

param fd_type
No help available

set_foffset(*foffset: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FOFFset
driver.source.bb.lora.impairments.set_foffset(foffset = 1)
```

No command help available

param foffset
No help available

set_st_error(*st_error: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:STERror
driver.source.bb.lora.impairments.set_st_error(st_error = 1)
```

No command help available

param st_error
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:STATe
driver.source.bb.lora.impairments.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.lora.impairments.clone()
```

Subgroups

6.18.3.19.3.1 Fdrift

SCPI Command :

```
[SOURce<HW>]:BB:LORA:IMPairments:FDRift:STATe
```

class FdriftCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDRift:STATe
value: bool = driver.source.bb.lora.impairments.fdrift.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:IMPairments:FDRift:STATe
driver.source.bb.lora.impairments.fdrift.set_state(state = False)
```

No command help available

param state
No help available

6.18.3.19.4 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:SETTing:CATalog
[SOURce<HW>]:BB:LORA:SETTing:DELete
[SOURce<HW>]:BB:LORA:SETTing:LOAD
```

class SettingCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:CATalog
value: List[str] = driver.source.bb.lora.setting.get_catalog()
```

No command help available

return
catalog: No help available

get_delete() → str

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:DELete
value: str = driver.source.bb.lora.setting.get_delete()
```

No command help available

return
filename: No help available

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:LOAD
value: str = driver.source.bb.lora.setting.get_load()
```

No command help available

return
filename: No help available

set_delete(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:DELete
driver.source.bb.lora.setting.set_delete(filename = 'abc')
```

No command help available

param filename
No help available

set_load(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:LOAD
driver.source.bb.lora.setting.set_load(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.bb.lora.setting.clone()
```

Subgroups

6.18.3.19.4.1 Store

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:SETTing:STORe:FAST
[SOURce<HW>]:BB:LORA:SETTing:STORe
```

class StoreCls

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

get_fast() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:STORe:FAST
value: bool = driver.source.bb.lora.setting.store.get_fast()
```

No command help available

return
fast: No help available

set_fast(*fast: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:STORe:FAST
driver.source.bb.lora.setting.store.set_fast(fast = False)
```

No command help available

param fast
No help available

set_value(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:SETTing:STORe
driver.source.bb.lora.setting.store.set_value(filename = 'abc')
```

No command help available

param filename
No help available

6.18.3.19.5 SymbolRate

SCPI Command :

```
[SOURce<HW>]:BB:LORA:SRATe:VARiation
```

class SymbolRateCls

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

get_variation() → float

```
# SCPI: [SOURce<HW>]:BB:LORA:SRATe:VARiation
value: float = driver.source.bb.lora.symbolRate.get_variation()
```

No command help available

return
variation: No help available

set_variation(*variation: float*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:SRATe:VARiation
driver.source.bb.lora.symbolRate.set_variation(variation = 1.0)
```

No command help available

param variation
No help available

6.18.3.19.6 Trigger

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:TRIGger:RMODe
[SOURce<HW>]:BB:LORA:TRIGger:SLENgth
[SOURce<HW>]:BB:LORA:TRIGger:SLUNit
[SOURce<HW>]:BB:LORA:TRIGger:SOURce
[SOURce<HW>]:BB:LORA:[TRIGger]:SEQuence
```

class TriggerCls

Trigger commands group definition. 22 total commands, 5 Subgroups, 5 group commands

get_rmode() → TrigRunMode

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:RMODe
value: enums.TrigRunMode = driver.source.bb.lora.trigger.get_rmode()
```

No command help available

return
rmode: No help available

get_sequence() → DmTrigMode

```
# SCPI: [SOURce<HW>]:BB:LORA:[TRIGger]:SEQuence
value: enums.DmTrigMode = driver.source.bb.lora.trigger.get_sequence()
```

No command help available

return
sequence: No help available

get_sl_unit() → UnitSlB

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:SLUNit
value: enums.UnitSlB = driver.source.bb.lora.trigger.get_sl_unit()
```

No command help available

return
sl_unit: No help available

get_slength() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:SLENgth
value: int = driver.source.bb.lora.trigger.get_slength()
```

No command help available

return
slength: No help available

get_source() → TriggerSourceB

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:SOURce
value: enums.TriggerSourceB = driver.source.bb.lora.trigger.get_source()
```

No command help available

return
source: No help available

set_rmode(*rmode: TrigRunMode*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:RMODe
driver.source.bb.lora.trigger.set_rmode(rmode = enums.TrigRunMode.RUN)
```

No command help available

param rmode
No help available

set_sequence(*sequence: DmTrigMode*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:[TRIGger]:SEQuence
driver.source.bb.lora.trigger.set_sequence(sequence = enums.DmTrigMode.AAUTo)
```

No command help available

param sequence
No help available

set_sl_unit(*sl_unit: UnitSlB*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:SLUNit
driver.source.bb.lora.trigger.set_sl_unit(sl_unit = enums.UnitSlB.SAMPle)
```

No command help available

param sl_unit
No help available

set_slength(*slength: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:SLENgth
driver.source.bb.lora.trigger.set_slength(slength = 1)
```

No command help available

param slength
No help available

set_source(*source: TriggerSourceB*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:SOURce
driver.source.bb.lora.trigger.set_source(source = enums.TriggerSourceB.
↪BEXTernal)
```

No command help available

param source
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.lora.trigger.clone()
```

Subgroups

6.18.3.19.6.1 Arm

class ArmCls

Arm 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.bb.lora.trigger.arm.clone()
```

Subgroups

6.18.3.19.6.2 Execute

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:ARM:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:ARM:EXECute
driver.source.bb.lora.trigger.arm.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:ARM:EXECute
driver.source.bb.lora.trigger.arm.execute.set_with_opc()
```

No command help available

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

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

6.18.3.19.6.3 Execute

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:EXECute
driver.source.bb.lora.trigger.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:EXECute
driver.source.bb.lora.trigger.execute.set_with_opc()
```

No command help available

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

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

6.18.3.19.6.4 External<External>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.source.bb.lora.trigger.external.repcap_external_get()
driver.source.bb.lora.trigger.external.repcap_external_set(repcap.External.Nr1)
```

class ExternalCls

External commands group definition. 3 total commands, 3 Subgroups, 0 group commands Repeated Capability: External, default value after init: External.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.lora.trigger.external.clone()
```

Subgroups

6.18.3.19.6.5 Delay

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:[EXTernal<CH>]:DELay
```

class DelayCls

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

get(*external=External.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:[EXTernal<CH>]:DELay
value: float = driver.source.bb.lora.trigger.external.delay.get(external =␣
↪repcap.External.Default)
```

No command help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

return
delay: No help available

set(*delay: float*, *external=External.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:[EXTernal<CH>]:DELay
driver.source.bb.lora.trigger.external.delay.set(delay = 1.0, external = repcap.
↪External.Default)
```

No command help available

param delay
No help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

6.18.3.19.6.6 Inhibit

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:[EXTernal<CH>]:INHibit
```

class InhibitCls

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

get(*external=External.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:[EXTernal<CH>]:INHibit
value: int = driver.source.bb.lora.trigger.external.inhibit.get(external =␣
↪repcap.External.Default)
```

No command help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

return
inhibit: No help available

set(*inhibit: int*, *external=External.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:[EXTernal<CH>]:INHibit
driver.source.bb.lora.trigger.external.inhibit.set(inhibit = 1, external =␣
↪repcap.External.Default)
```

No command help available

param inhibit
No help available

param external
optional repeated capability selector. Default value: Nr1 (settable in the interface 'External')

6.18.3.19.6.7 Synchronize

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:[EXTernal]:SYNChronize:OUTPut
```

class SynchronizeCls

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

get_output() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:[EXTernal]:SYNChronize:OUTPut
value: bool = driver.source.bb.lora.trigger.external.synchronize.get_output()
```

No command help available

return
output: No help available

set_output(*output: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:[EXTernal]:SYNChronize:OUTPut
driver.source.bb.lora.trigger.external.synchronize.set_output(output = False)
```

No command help available

param output
No help available

6.18.3.19.6.8 Obaseband

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:TRIGger:OBASeband:DELay
[SOURce<HW>]:BB:LORA:TRIGger:OBASeband:INHibit
```

class ObasebandCls

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

get_delay() → float

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OBASeband:DELay
value: float = driver.source.bb.lora.trigger.obaseband.get_delay()
```

No command help available

return
delay: No help available

get_inhibit() → int

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OBASeband:INHibit
value: int = driver.source.bb.lora.trigger.obaseband.get_inhibit()
```

No command help available

return
inhibit: No help available

set_delay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OBASeband:DELay
driver.source.bb.lora.trigger.obaseband.set_delay(delay = 1.0)
```

No command help available

param delay
No help available

set_inhibit(*inhibit: int*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OBASeband:INHibit
driver.source.bb.lora.trigger.obaseband.set_inhibit(inhibit = 1)
```

No command help available

param inhibit
No help available

6.18.3.19.6.9 Output<Output>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.lora.trigger.output.repcap_output_get()
driver.source.bb.lora.trigger.output.repcap_output_set(repcap.Output.Nr1)
```

class OutputCls

Output commands group definition. 10 total commands, 6 Subgroups, 0 group commands Repeated Capability: Output, default value after init: Output.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.lora.trigger.output.clone()
```

Subgroups

6.18.3.19.6.10 Delay

SCPI Commands :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:DELay
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut:DELay:FIXed
```

class DelayCls

Delay commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:DELay
value: float = driver.source.bb.lora.trigger.output.delay.get(output = repcap.
↪Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
delay: No help available

get_fixed() → bool

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut:DELay:FIXed
value: bool = driver.source.bb.lora.trigger.output.delay.get_fixed()
```

No command help available

return
fixed: No help available

set(*delay: float*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:DELay
driver.source.bb.lora.trigger.output.delay.set(delay = 1.0, output = repcap.
↪Output.Default)
```

No command help available

param delay
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

set_fixed(*fixed: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut:DELay:FIXed
driver.source.bb.lora.trigger.output.delay.set_fixed(fixed = False)
```

No command help available

param fixed
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.lora.trigger.output.delay.clone()
```

Subgroups

6.18.3.19.6.11 Maximum

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:DELay:MAXimum
```

class MaximumCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:DELay:MAXimum
value: float = driver.source.bb.lora.trigger.output.delay.maximum.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
maximum: No help available

6.18.3.19.6.12 Minimum

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:DELay:MINimum
```

class MinimumCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:DELay:MINimum
value: float = driver.source.bb.lora.trigger.output.delay.minimum.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
minimum: No help available

6.18.3.19.6.13 Mode

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:MODE
```

class ModeCls

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

get(*output=Output.Default*) → MarkModeA

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:MODE
value: enums.MarkModeA = driver.source.bb.lora.trigger.output.mode.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
mode: No help available

set(*mode: MarkModeA*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:MODE
driver.source.bb.lora.trigger.output.mode.set(mode = enums.MarkModeA.FRAMe,␣
↪output = repcap.Output.Default)
```

No command help available

param mode
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.19.6.14 OffTime

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:OFFTime
```

class OffTimeCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:OFFTime
value: int = driver.source.bb.lora.trigger.output.offTime.get(output = repcap.
↪Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
off_time: No help available

set(*off_time: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:OFFTime
driver.source.bb.lora.trigger.output.offTime.set(off_time = 1, output = repcap.
↪Output.Default)
```

No command help available

param off_time
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.19.6.15 Ontime

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:ONTime
```

class OntimeCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:ONTime
value: int = driver.source.bb.lora.trigger.output.ontime.get(output = repcap.
↪Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
ontime: No help available

set(*ontime: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:ONTime
driver.source.bb.lora.trigger.output.ontime.set(ontime = 1, output = repcap.
↪Output.Default)
```

No command help available

param ontime
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.19.6.16 Pattern

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PATTern
```

class PatternCls

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

class PatternStruct

Response structure. Fields:

- Pattern: List[str]: No parameter help available
- Bitcount: int: No parameter help available

get(*output=Output.Default*) → PatternStruct

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PATTern
value: PatternStruct = driver.source.bb.lora.trigger.output.pattern.get(output␣
↪= repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
structure: for return value, see the help for PatternStruct structure arguments.

set(*pattern: List[str]*, *bitcount: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PATTern
driver.source.bb.lora.trigger.output.pattern.set(pattern = ['rawAbc1', 'rawAbc2
↪', 'rawAbc3'], bitcount = 1, output = repcap.Output.Default)
```

No command help available

param pattern
No help available

param bitcount
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.19.6.17 Pulse

class PulseCls

Pulse 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.bb.lora.trigger.output.pulse.clone()
```

Subgroups

6.18.3.19.6.18 Divider

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PULSe:DIVider
```

class DividerCls

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

get(*output=Output.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PULSe:DIVider
value: int = driver.source.bb.lora.trigger.output.pulse.divider.get(output =␣
↪repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
divider: No help available

set(*divider: int*, *output=Output.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PULSe:DIVider
driver.source.bb.lora.trigger.output.pulse.divider.set(divider = 1, output =␣
↪repcap.Output.Default)
```

No command help available

param divider
No help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

6.18.3.19.6.19 Frequency

SCPI Command :

```
[SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PULSe:FREQuency
```

class FrequencyCls

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

get(*output=Output.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:LORA:TRIGger:OUTPut<CH>:PULSe:FREQuency
value: float = driver.source.bb.lora.trigger.output.pulse.frequency.get(output␣
↪= repcap.Output.Default)
```

No command help available

param output
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Output')

return
frequency: No help available

6.18.3.19.7 Waveform

SCPI Command :

```
[SOURce<HW>]:BB:LORA:WAVeform:CREate
```

class WaveformCls

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

set_create(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:BB:LORA:WAVeform:CREate
driver.source.bb.lora.waveform.set_create(filename = 'abc')
```

No command help available

param filename
No help available

6.18.3.20 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

6.18.3.21 Power

SCPI Commands :

```
[SOURce<HW>]:BB:POWer:PEAK
[SOURce<HW>]:BB:POWer:RMS
```

class PowerCls

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

get_peak() → float

```
# SCPI: [SOURce<HW>]:BB:POWer:PEAK
value: float = driver.source.bb.power.get_peak()
```

Queries the peak level of the baseband signal relative to full scale of 0.5 V (in terms of dB full scale) .

return
peak: float Range: -145 to 30, Unit: dBfs

get_rms() → float

```
# SCPI: [SOURce<HW>]:BB:POWer:RMS
value: float = driver.source.bb.power.get_rms()
```

Queries the RMS level of the baseband signal relative to full scale of 0.5V (in terms of dB full scale) .

return
rms: float Range: -145 to 30, Unit: dBfs

6.18.3.22 Progress

class ProgressCls

Progress 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.source.bb.progress.clone()
```

Subgroups

6.18.3.22.1 Mcoder

SCPI Command :

```
[SOURce<HW>]:BB:PROGress:MCODer
```

class McoderCls

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

get_value() → int

```
# SCPI: [SOURce<HW>]:BB:PROGress:MCODer
value: int = driver.source.bb.progress.mcoder.get_value()
```

Queries the status of an initiated process, like for example the calculation of a signal in accordance to a digital standard, or the calculation of a multi-carrier or multi-segment waveform file.

return
mcoder: integer Indicates the task progress in percent Range: 0 to 100

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.progress.mcoder.clone()
```

Subgroups

6.18.3.22.1.1 Arbitrary

SCPI Commands :

```
[SOURce<HW>]:BB:PROGress:MCODer:ARBitrary:MCARrier
[SOURce<HW>]:BB:PROGress:MCODer:ARBitrary:WSEGment
```

class ArbitraryCls

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

get_mcarrier() → int

```
# SCPI: [SOURce<HW>]:BB:PROGress:MCODer:ARBitrary:MCARrier
value: int = driver.source.bb.progress.mcoder.arbitrary.get_mcarrier()
```

Queries the status of an initiated process, like for example the calculation of a signal in accordance to a digital standard, or the calculation of a multi-carrier or multi-segment waveform file.

return
mcarrier: integer Indicates the task progress in percent Range: 0 to 100

get_wsegment() → int

```
# SCPI: [SOURce<HW>]:BB:PROGress:MCODer:ARBitrary:WSEGment
value: int = driver.source.bb.progress.mcoder.arbitrary.get_wsegment()
```

Queries the status of an initiated process, like for example the calculation of a signal in accordance to a digital standard, or the calculation of a multi-carrier or multi-segment waveform file.

return
wsegment: integer Indicates the task progress in percent Range: 0 to 100

6.18.3.23 Radio

class RadioCls

Radio commands group definition. 183 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.18.3.23.1 Am

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:AM:DEPTh
[SOURce<HW>]:BB:RADio:AM:INPut
[SOURce<HW>]:BB:RADio:AM:PRESet
[SOURce<HW>]:BB:RADio:AM:SOURce
[SOURce<HW>]:BB:RADio:AM:STATe
```

class AmCls

Am commands group definition. 16 total commands, 5 Subgroups, 5 group commands

get_depth() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:DEPTh
value: float = driver.source.bb.radio.am.get_depth()
```

Sets the nominal modulation depth.

return
depth: float Range: 0 to 100

get_input_py() → AudioBcInputSignal

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:INPut
value: enums.AudioBcInputSignal = driver.source.bb.radio.am.get_input_py()
```

Sets the audio source for the AM modulator signal.

return
input_py: EXTernal| AGENerator| APLayer| OFF EXTernal Uses an external audio signal input at the 'User 2' connector. The audio source is fixed to 'Source S/PDIF', see [:SOURcehw]:BB:RADio:AM:SOURce. AGENerator Uses an internal audio generator as the signal source. APLayer Uses an audio player file, that is saved to the memory of the R&S SMCV100B. OFF Disables the audio source for the AM modulator.

get_source() → BcInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SOURce
value: enums.BcInputSignalSource = driver.source.bb.radio.am.get_source()
```

Queries the audio source.

return
source: SPDif is fixed.

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:STATe
value: bool = driver.source.bb.radio.am.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

preset() → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:PRESet
driver.source.bb.radio.am.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>:BB:AM|FM|FM:RDS:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:PRESet
driver.source.bb.radio.am.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>:BB:AM|FM|FM:RDS:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_depth(*depth: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:DEPTh
driver.source.bb.radio.am.set_depth(depth = 1.0)
```

Sets the nominal modulation depth.

param depth
float Range: 0 to 100

set_input_py(*input_py: AudioBcInputSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:INPut
driver.source.bb.radio.am.set_input_py(input_py = enums.AudioBcInputSignal.
↪AGENerator)
```

Sets the audio source for the AM modulator signal.

param input_py
EXTernal| AGENerator| APLayer| OFF EXTernal Uses an external audio signal input at the 'User 2' connector. The audio source is fixed to 'Source S/PDIF', see

[:SOURcehw]:BB:RADio:AM:SOURce. AGENerator Uses an internal audio generator as the signal source. APLayer Uses an audio player file, that is saved to the memory of the R&S SMCV100B. OFF Disables the audio source for the AM modulator.

set_source(*source: BcInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SOURce
driver.source.bb.radio.am.set_source(source = enums.BcInputSignalSource.SPDif)
```

Queries the audio source.

param source
SPDif is fixed.

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:STATe
driver.source.bb.radio.am.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.am.clone()
```

Subgroups

6.18.3.23.1.1 ApLayer

SCPI Command :

```
[SOURce<HW>]:BB:RADio:AM:APLayer:ATT
```

class ApLayerCls

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

get_att() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:APLayer:ATT
value: float = driver.source.bb.radio.am.apLayer.get_att()
```

Sets the attenuation.

return
attenuation: float Range: 0 to 30

set_att(*attenuation: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:APLayer:ATT
driver.source.bb.radio.am.apLayer.set_att(attenuation = 1.0)
```

Sets the attenuation.

param attenuation
float Range: 0 to 30

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.am.apLayer.clone()
```

Subgroups

6.18.3.23.1.2 Library

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:AM:APLayer:LIBRary:CATalog
[SOURce<HW>]:BB:RADio:AM:APLayer:LIBRary:SELect
```

class LibraryCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:APLayer:LIBRary:CATalog
value: List[str] = driver.source.bb.radio.am.apLayer.library.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.wv and *.wav. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
tx_audio_bc_am_wav_cat_nam: No help available

get_select() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:APLayer:LIBRary:SELect
value: str = driver.source.bb.radio.am.apLayer.library.get_select()
```

Selects the audio file. If no file of the specified name exists, an error message is displayed. You can select files with the file extension *.wv and *.wav. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
sel: string Filename or complete file path; file extension can be omitted

set_select(*sel: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:APLayer:LIBRary:SELect
driver.source.bb.radio.am.apLayer.library.set_select(sel = 'abc')
```

Selects the audio file. If no file of the specified name exists, an error message is displayed. You can select files with the file extension *.wv and *.wav. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param sel
string Filename or complete file path; file extension can be omitted

6.18.3.23.1.3 AudGen

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:AM:AUDGen:FRQ
[SOURce<HW>]:BB:RADio:AM:AUDGen:LEV
```

class AudGenCls

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

get_frq() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:AUDGen:FRQ
value: float = driver.source.bb.radio.am.audGen.get_frq()
```

Sets the frequency.

return
freq: float Range: 0.03 to 15, Unit: kHz

get_lev() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:AUDGen:LEV
value: float = driver.source.bb.radio.am.audGen.get_lev()
```

Sets the level.

return
level: float Range: -60 to 12, Unit: dBu

set_frq(*freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:AUDGen:FRQ
driver.source.bb.radio.am.audGen.set_frq(freq = 1.0)
```

Sets the frequency.

param freq
float Range: 0.03 to 15, Unit: kHz

set_lev(*level: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:AUDGen:LEV
driver.source.bb.radio.am.audGen.set_lev(level = 1.0)
```

Sets the level.

param level
float Range: -60 to 12, Unit: dBu

6.18.3.23.1.4 Audio

SCPI Command :

```
[SOURce<HW>]:BB:RADio:AM:AUDio:AF
```

class AudioCls

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

get_af() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:AUDio:AF
value: bool = driver.source.bb.radio.am.audio.get_af()
```

Enables or disables the audio channel.

return
audio: 1| ON| 0| OFF

set_af(*audio: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:AUDio:AF
driver.source.bb.radio.am.audio.set_af(audio = False)
```

Enables or disables the audio channel.

param audio
1| ON| 0| OFF

6.18.3.23.1.5 Modulation

SCPI Command :

```
[SOURce<HW>]:BB:RADio:AM:MODulation:DEPTh
```

class ModulationCls

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

get_depth() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:MODulation:DEPTh
value: int = driver.source.bb.radio.am.modulation.get_depth()
```

Displays the modulation depth.

return
mod_depth: integer Range: 0 to 100

6.18.3.23.1.6 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:AM:SETTing:CATalog
[SOURce<HW>]:BB:RADio:AM:SETTing:DELete
[SOURce<HW>]:BB:RADio:AM:SETTing:LOAD
[SOURce<HW>]:BB:RADio:AM:SETTing:STORe
```

class SettingCls

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

delete(*am_del: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SETTing:DELete
driver.source.bb.radio.am.setting.delete(am_del = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param am_del
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SETTing:CATalog
value: List[str] = driver.source.bb.radio.am.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
tx_audio_bc_am_cat_name: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SETTing:LOAD
value: str = driver.source.bb.radio.am.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
am_rcl: 'filename' Filename or complete file path; file extension can be omitted

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SETTing:STORe
value: str = driver.source.bb.radio.am.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.am/fm/rds) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
am_sav: 'filename' Filename or complete file path

set_load(*am_rcl: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SETTing:LOAD
driver.source.bb.radio.am.setting.set_load(am_rcl = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param am_rcl
'filename' Filename or complete file path; file extension can be omitted

set_store(*am_sav: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:AM:SETTing:STORe
driver.source.bb.radio.am.setting.set_store(am_sav = 'abc')
```

Saves the current settings into the selected file; the file extension (*.am/fm/rds) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param am_sav
'filename' Filename or complete file path

6.18.3.23.2 Fm

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:INPut
[SOURce<HW>]:BB:RADio:FM:MODE
[SOURce<HW>]:BB:RADio:FM:PRESet
[SOURce<HW>]:BB:RADio:FM:STATe
```

class FmCls

Fm commands group definition. 167 total commands, 8 Subgroups, 4 group commands

get_input_py() → AudioBcInputSignal

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:INPut
value: enums.AudioBcInputSignal = driver.source.bb.radio.fm.get_input_py()
```

Sets the audio source for the FM modulator signal.

return
input_py: EXTernal| AGENerator| APLayer| OFF EXTernal Uses an external audio signal input at the 'User 2' connector. The audio source is fixed to 'Source S/PDIF', see [:SOURcehw]:BB:RADio:FM:AUDio:SOURce?. AGENerator Uses an internal audio generator as the signal source. APLayer Uses an audio player file, that is saved to the memory of the R&S SMCV100B. OFF Disables the audio source for the FM modulator.

get_mode() → AudioBcFmModulationMode

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:MODE
value: enums.AudioBcFmModulationMode = driver.source.bb.radio.fm.get_mode()
```

Sets the mode.

return
mode: MONO| STEReo MONO Feeds a mono signal to the modulator with band limitation 15 kHz. STEReo Feeds a stereo signal to the modulator.

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:STATe
value: bool = driver.source.bb.radio.fm.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

preset() → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:PRESet
driver.source.bb.radio.fm.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>:BB:AM|FM|FM:RDS:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:PRESet
driver.source.bb.radio.fm.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>:BB:AM|FM|FM:RDS:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_input_py(*input_py: AudioBcInputSignal*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:INPut
driver.source.bb.radio.fm.set_input_py(input_py = enums.AudioBcInputSignal.
↪AGENerator)
```

Sets the audio source for the FM modulator signal.

param input_py
EXTernal| AGENerator| APLayer| OFF EXTernal Uses an external audio signal input at the 'User 2' connector. The audio source is fixed to 'Source S/PDIF', see [:SOURcehw]:BB:RADio:FM:AUDio:SOURce?. AGENerator Uses an internal audio generator as the signal source. APLayer Uses an audio player file, that is saved to the memory of the R&S SMCV100B. OFF Disables the audio source for the FM modulator.

set_mode(*mode: AudioBcFmModulationMode*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:MODE
driver.source.bb.radio.fm.set_mode(mode = enums.AudioBcFmModulationMode.MONO)
```

Sets the mode.

param mode
MONO| STEReo MONO Feeds a mono signal to the modulator with band limitation 15 kHz. STEReo Feeds a stereo signal to the modulator.

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:STATe
driver.source.bb.radio.fm.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.clone()
```

Subgroups

6.18.3.23.2.1 ApLayer

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:APLayer:ATT1
[SOURce<HW>]:BB:RADio:FM:APLayer:ATT2
```

class ApLayerCls

ApLayer commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_att_1() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:APLayer:ATT1
value: float = driver.source.bb.radio.fm.apLayer.get_att_1()
```

Sets the attenuation.

return
attl: No help available

get_att_2() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:APLayer:ATT2
value: float = driver.source.bb.radio.fm.apLayer.get_att_2()
```

Sets the attenuation.

return
attr: float Range: 0 to 30, Unit: dB

set_att_1(*attl: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:APLayer:ATT1
driver.source.bb.radio.fm.apLayer.set_att_1(attl = 1.0)
```

Sets the attenuation.

param attl
float Range: 0 to 30, Unit: dB

set_att_2(*attr: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:APLayer:ATT2
driver.source.bb.radio.fm.apLayer.set_att_2(attr = 1.0)
```

Sets the attenuation.

param attr
float Range: 0 to 30, Unit: dB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.apLayer.clone()
```

Subgroups

6.18.3.23.2.2 Library

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:APLayer:LIBRary:CATalog
[SOURce<HW>]:BB:RADio:FM:APLayer:LIBRary:SELect
```

class LibraryCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:APLayer:LIBRary:CATalog
value: List[str] = driver.source.bb.radio.fm.apLayer.library.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.wv and *.wav. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
tx_audio_bc_fm_wav_cat_nam: No help available

get_select() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:APLayer:LIBRary:SELect
value: str = driver.source.bb.radio.fm.apLayer.library.get_select()
```

Selects the audio file. If no file of the specified name exists, an error message is displayed. You can select files with the file extension *.wv and *.wav. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
sel: string Filename or complete file path; file extension can be omitted

set_select(*sel: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:APLayer:LIBRary:SELect
driver.source.bb.radio.fm.apLayer.library.set_select(sel = 'abc')
```

Selects the audio file. If no file of the specified name exists, an error message is displayed. You can select files with the file extension *.wv and *.wav. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param sel
string Filename or complete file path; file extension can be omitted

6.18.3.23.2.3 AudGen

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:AUDGen:FRQ1
[SOURce<HW>]:BB:RADio:FM:AUDGen:FRQ2
[SOURce<HW>]:BB:RADio:FM:AUDGen:LEV1
[SOURce<HW>]:BB:RADio:FM:AUDGen:LEV2
```

class AudGenCls

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

get_frq_1() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:FRQ1
value: int = driver.source.bb.radio.fm.audGen.get_frq_1()
```

Sets the frequency.

return
freq_left: No help available

get_frq_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:FRQ2
value: int = driver.source.bb.radio.fm.audGen.get_frq_2()
```

Sets the frequency.

return
freq_right: integer Range: 30 to 15000, Unit: Hz

get_lev_1() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:LEV1
value: float = driver.source.bb.radio.fm.audGen.get_lev_1()
```

Sets the level.

return
level_left: No help available

get_lev_2() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:LEV2
value: float = driver.source.bb.radio.fm.audGen.get_lev_2()
```

Sets the level.

return
level_right: float Range: -60 to 12, Unit: dBu

set_frq_1(*freq_left: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:FRQ1
driver.source.bb.radio.fm.audGen.set_frq_1(freq_left = 1)
```

Sets the frequency.

param freq_left
integer Range: 30 to 15000, Unit: Hz

set_frq_2(*freq_right: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:FRQ2
driver.source.bb.radio.fm.audGen.set_frq_2(freq_right = 1)
```

Sets the frequency.

param freq_right
integer Range: 30 to 15000, Unit: Hz

set_lev_1(*level_left: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:LEV1
driver.source.bb.radio.fm.audGen.set_lev_1(level_left = 1.0)
```

Sets the level.

param level_left
float Range: -60 to 12, Unit: dBu

set_lev_2(*level_right: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDGen:LEV2
driver.source.bb.radio.fm.audGen.set_lev_2(level_right = 1.0)
```

Sets the level.

param level_right
float Range: -60 to 12, Unit: dBu

6.18.3.23.2.4 Audio

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:AUDio:AF1
[SOURce<HW>]:BB:RADio:FM:AUDio:AF2
[SOURce<HW>]:BB:RADio:FM:AUDio:DEViation
[SOURce<HW>]:BB:RADio:FM:AUDio:MODE
[SOURce<HW>]:BB:RADio:FM:AUDio:NDEViation
[SOURce<HW>]:BB:RADio:FM:AUDio:PREemphasis
[SOURce<HW>]:BB:RADio:FM:AUDio:SOURce
```

class AudioCls

Audio commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_af_1() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:AF1
value: bool = driver.source.bb.radio.fm.audio.get_af_1()
```

Enables or disables the audio channel.

return
audio_l: No help available

get_af_2() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:AF2
value: bool = driver.source.bb.radio.fm.audio.get_af_2()
```

Enables or disables the audio channel.

return
audio_r: 1| ON| 0| OFF

get_deviation() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:DEViation
value: float = driver.source.bb.radio.fm.audio.get_deviation()
```

Queries the actual frequency deviation.

return
freq_dev_audio: float Range: 0 to 999.99, Unit: kHz

get_mode() → AudioBcFmInputSignalAfMode

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:MODE
value: enums.AudioBcFmInputSignalAfMode = driver.source.bb.radio.fm.audio.get_
↪mode()
```

Sets the relationship of the two audio channels with respect to each other.

return
af_mode: LEFT| RIGHt| RELeft| REMLeft| RNELeft

get_ndeviation() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:NDEViation
value: float = driver.source.bb.radio.fm.audio.get_ndeviation()
```

Defines the signal deviation, that is the deviation only caused by the audio signals.

return
mon_freq_dev_audio: float Range: 0 to 100, Unit: kHz

get_preemphasis() → AudioBcFmModulationPreemphasis

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:PREemphasis
value: enums.AudioBcFmModulationPreemphasis = driver.source.bb.radio.fm.audio.
↪get_preemphasis()
```

Sets the preemphasis factor for the signal to noise ratio improvement.

return
preemphasis: OFF| D50us| D75us

get_source() → BcInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:SOURce
value: enums.BcInputSignalSource = driver.source.bb.radio.fm.audio.get_source()
```

Queries the audio source.

return
source: SPDif is fixed.

set_af_1(*audio_l: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:AF1
driver.source.bb.radio.fm.audio.set_af_1(audio_l = False)
```

Enables or disables the audio channel.

param audio_l
1| ON| 0| OFF

set_af_2(*audio_r: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:AF2
driver.source.bb.radio.fm.audio.set_af_2(audio_r = False)
```

Enables or disables the audio channel.

param audio_r
1| ON| 0| OFF

set_mode(*af_mode: AudioBcFmInputSignalAfMode*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:MODE
driver.source.bb.radio.fm.audio.set_mode(af_mode = enums.
↪AudioBcFmInputSignalAfMode.LEFT)
```

Sets the relationship of the two audio channels with respect to each other.

param af_mode
LEFT| RIGHt| RELeft| REMLeft| RNELeft

set_ndeviation(*mon_freq_dev_audio: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:NDEViation
driver.source.bb.radio.fm.audio.set_ndeviation(mon_freq_dev_audio = 1.0)
```

Defines the signal deviation, that is the deviation only caused by the audio signals.

param mon_freq_dev_audio
float Range: 0 to 100, Unit: kHz

set_preemphasis(*preemphasis: AudioBcFmModulationPreemphasis*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:AUDio:PREemphasis
driver.source.bb.radio.fm.audio.set_preemphasis(preemphasis = enums.
↪AudioBcFmModulationPreemphasis.D50us)
```

Sets the preemphasis factor for the signal to noise ratio improvement.

param preemphasis
OFF| D50us| D75us

6.18.3.23.2.5 Darc

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:DARC:DEViation
[SOURce<HW>]:BB:RADio:FM:DARC:INFormation
[SOURce<HW>]:BB:RADio:FM:DARC:[STATe]
```

class DarcCls

Darc commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_deviation() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:DEViation
value: float = driver.source.bb.radio.fm.darc.get_deviation()
```

No command help available

return
freq_dev_darc: float Range: 0 to 10

get_information() → AudioBcFmDarcInformation

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:INFormation
value: enums.AudioBcFmDarcInformation = driver.source.bb.radio.fm.darc.get_
↪information()
```

No command help available

return
darc_inf: OFF| PRBS| DATa

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:[STATe]
value: bool = driver.source.bb.radio.fm.darc.get_state()
```

No command help available

return
darc: 1| ON| 0| OFF

set_deviation(*freq_dev_darc: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:DEViation
driver.source.bb.radio.fm.darc.set_deviation(freq_dev_darc = 1.0)
```

No command help available

param freq_dev_darc
float Range: 0 to 10

set_information(*darc_inf: AudioBcFmDarcInformation*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:INFormation
driver.source.bb.radio.fm.darc.set_information(darc_inf = enums.
↪AudioBcFmDarcInformation.DATa)
```

No command help available

param darc_inf
OFF| PRBS| DATa

set_state(*darc: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:[STATe]
driver.source.bb.radio.fm.darc.set_state(darc = False)
```

No command help available

param darc
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.darc.clone()
```

Subgroups

6.18.3.23.2.6 Bic<BlockIdCode>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.source.bb.radio.fm.darc.bic.repcap_blockIdCode_get()
driver.source.bb.radio.fm.darc.bic.repcap_blockIdCode_set(repcap.BlockIdCode.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:DARC:BIC<CH>
```

class BicCls

Bic commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: BlockIdCode, default value after init: BlockIdCode.Nr1

get(*blockIdCode=BlockIdCode.Default*) → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:BIC<CH>
value: str = driver.source.bb.radio.fm.darc.bic.get(blockIdCode = repcap.
↪BlockIdCode.Default)
```

Specifies data for block identification codes 1 to 3.

param blockIdCode
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Bic')

return
darc_bic: string

set(*darc_bic: str*, *blockIdCode=BlockIdCode.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:DARC:BIC<CH>
driver.source.bb.radio.fm.darc.bic.set(darc_bic = 'abc', blockIdCode = repcap.
↪BlockIdCode.Default)
```

Specifies data for block identification codes 1 to 3.

param darc_bic
string

param blockIdCode
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Bic')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.darc.bic.clone()
```

6.18.3.23.2.7 Pilot

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:PILot:DEViation
```

class PilotCls

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

get_deviation() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:PILot:DEViation
value: float = driver.source.bb.radio.fm.pilot.get_deviation()
```

Defines the resulting 19 kHz frequency deviation of the pilot tone irrespective of the audio signals.

return
freq_dev_pilot: float Range: 0 to 15, Unit: kHz

set_deviation(*freq_dev_pilot: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:PILot:DEViation
driver.source.bb.radio.fm.pilot.set_deviation(freq_dev_pilot = 1.0)
```

Defines the resulting 19 kHz frequency deviation of the pilot tone irrespective of the audio signals.

param freq_dev_pilot
float Range: 0 to 15, Unit: kHz

6.18.3.23.2.8 Rds

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:CT
[SOURce<HW>]:BB:RADio:FM:RDS:CTOFfset
[SOURce<HW>]:BB:RADio:FM:RDS:DEViation
[SOURce<HW>]:BB:RADio:FM:RDS:MS
[SOURce<HW>]:BB:RADio:FM:RDS:PI
[SOURce<HW>]:BB:RADio:FM:RDS:PS
[SOURce<HW>]:BB:RADio:FM:RDS:PTY
[SOURce<HW>]:BB:RADio:FM:RDS:PTYN
[SOURce<HW>]:BB:RADio:FM:RDS:RT
[SOURce<HW>]:BB:RADio:FM:RDS:TA
[SOURce<HW>]:BB:RADio:FM:RDS:[STATe]
```

class RdsCls

Rds commands group definition. 135 total commands, 7 Subgroups, 11 group commands

get_ct() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:CT
value: bool = driver.source.bb.radio.fm.rds.get_ct()
```

Enables/disables the clock time and date information.

return
ct: 1| ON| 0| OFF

get_ct_offset() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:CTOFfset
value: str = driver.source.bb.radio.fm.rds.get_ct_offset()
```

Sets the clock time offset.

return
ct_offset: string Range: 00:00 to 99:59

get_deviation() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DEViation
value: float = driver.source.bb.radio.fm.rds.get_deviation()
```

Defines the resulting frequency deviation of the radio data system irrespective of the audio signals.

return
freq_dev_rds: float Range: 0 to 10, Unit: kHz

get_ms() → TxAudioBcFmRdsMs

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:MS
value: enums.TxAudioBcFmRdsMs = driver.source.bb.radio.fm.rds.get_ms()
```

Identifies if the transmission contains music or speech.

return
ms: MUSic| SPEech

get_pi() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PI
value: int = driver.source.bb.radio.fm.rds.get_pi()
```

Sets the program identification, that is a 16-bit value in hexadecimal representation.

return
pi: integer Range: #H0000 to #HFFFF

get_ps() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PS
value: str = driver.source.bb.radio.fm.rds.get_ps()
```

Sets the program service name.

return
ps: string Up to eight characters in ASCII format, see Figure 'Character sets for names'.

get_pty() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PTY
value: int = driver.source.bb.radio.fm.rds.get_pty()
```

Sets the program type.

return
pty: integer Range: 0 to 31

get_ptyn() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PTYN
value: str = driver.source.bb.radio.fm.rds.get_ptyn()
```

Sets the program type name.

return
ptyn: string Up to eight characters in ASCII format, see Figure 'Character sets for names'.

get_rt() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:RT
value: str = driver.source.bb.radio.fm.rds.get_rt()
```

Sets the radio text.

return
rt: string Up to 64 characters in ASCII format, see Figure 'Character sets for names'.

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:[STATe]
value: bool = driver.source.bb.radio.fm.rds.get_state()
```

Enables/disables /.

return
rds_state: 1| ON| 0| OFF

get_ta() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TA
value: bool = driver.source.bb.radio.fm.rds.get_ta()
```

Enables/disables the traffic announcement flag.

return
ta: 1| ON| 0| OFF

set_ct(*ct: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:CT
driver.source.bb.radio.fm.rds.set_ct(ct = False)
```

Enables/disables the clock time and date information.

param ct
1| ON| 0| OFF

set_ct_offset(*ct_offset: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:CTOFfset
driver.source.bb.radio.fm.rds.set_ct_offset(ct_offset = 'abc')
```

Sets the clock time offset.

param ct_offset
string Range: 00:00 to 99:59

set_deviation(*freq_dev_rds: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DEViation
driver.source.bb.radio.fm.rds.set_deviation(freq_dev_rds = 1.0)
```

Defines the resulting frequency deviation of the radio data system irrespective of the audio signals.

param freq_dev_rds
float Range: 0 to 10, Unit: kHz

set_ms(*ms: TxAudioBcFmRdsMs*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:MS
driver.source.bb.radio.fm.rds.set_ms(ms = enums.TxAudioBcFmRdsMs.MUSic)
```

Identifies if the transmission contains music or speech.

param ms
MUSic| SPEech

set_pi(*pi: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PI
driver.source.bb.radio.fm.rds.set_pi(pi = 1)
```

Sets the program identification, that is a 16-bit value in hexadecimal representation.

param pi
integer Range: #H0000 to #HFFFF

set_ps(*ps: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PS
driver.source.bb.radio.fm.rds.set_ps(ps = 'abc')
```

Sets the program service name.

param ps
string Up to eight characters in ASCII format, see Figure 'Character sets for names'.

set_pty(*pty: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PTY
driver.source.bb.radio.fm.rds.set_pty(pty = 1)
```

Sets the program type.

param pty
integer Range: 0 to 31

set_ptyn(*ptyn: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:PTYN
driver.source.bb.radio.fm.rds.set_ptyn(ptyn = 'abc')
```

Sets the program type name.

param ptyn
string Up to eight characters in ASCII format, see Figure 'Character sets for names'.

set_rt(*rt: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:RT
driver.source.bb.radio.fm.rds.set_rt(rt = 'abc')
```

Sets the radio text.

param rt
string Up to 64 characters in ASCII format, see Figure 'Character sets for names'.

set_state(*rds_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:[STATe]
driver.source.bb.radio.fm.rds.set_state(rds_state = False)
```

Enables/disables /.

param rds_state
1| ON| 0| OFF

set_ta(*ta: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TA
driver.source.bb.radio.fm.rds.set_ta(ta = False)
```

Enables/disables the traffic announcement flag.

param ta
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.clone()
```

Subgroups

6.18.3.23.2.9 Af

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:METHod
```

class AfCls

Af commands group definition. 23 total commands, 2 Subgroups, 1 group commands

get_method() → MappingType

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:METHod
value: enums.MappingType = driver.source.bb.radio.fm.rds.af.get_method()
```

No command help available

return
af_method: B| A

set_method(*af_method: MappingType*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:METHod
driver.source.bb.radio.fm.rds.af.set_method(af_method = enums.MappingType.A)
```

No command help available

param af_method
B| A

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.clone()
```

Subgroups

6.18.3.23.2.10 A

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:A:NUMBer
```

class ACls

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

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:A:NUMBer
value: int = driver.source.bb.radio.fm.rds.af.a.get_number()
```

Defines the number of alternative frequencies.

return
af_num_freq_a: integer Range: 0 to 25

set_number(*af_num_freq_a: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:A:NUMBer
driver.source.bb.radio.fm.rds.af.a.set_number(af_num_freq_a = 1)
```

Defines the number of alternative frequencies.

param af_num_freq_a
integer Range: 0 to 25

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.a.clone()
```

Subgroups

6.18.3.23.2.11 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.af.a.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.af.a.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:A:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:A:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.af.a.frequency.get(index = repcap.
↪Index.Default)
```

Sets the alternative frequencies in AF method A.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
af_freq_a: float Range: 87.6 to 107.9

set(*af_freq_a: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:A:FREQuency<CH>
driver.source.bb.radio.fm.rds.af.a.frequency.set(af_freq_a = 1.0, index =␣
↪repcap.Index.Default)
```

Sets the alternative frequencies in AF method A.

param af_freq_a
float Range: 87.6 to 107.9

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.a.frequency.clone()
```

6.18.3.23.2.12 B

class BCls

B commands group definition. 20 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.clone()
```

Subgroups

6.18.3.23.2.13 List1

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:NUMBer
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:TFRequency
```

class List1Cls

List1 commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:NUMBer
value: int = driver.source.bb.radio.fm.rds.af.b.list1.get_number()
```

Sets the number of frequencies of a list in AF method B.

return
afb_list_1_no_freq: No help available

get_tfrequency() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:TFRequency
value: float = driver.source.bb.radio.fm.rds.af.b.list1.get_tfrequency()
```

Sets the tuning frequency of a list in AF method B.

return
af_list_1_tun_freq: No help available

set_number(*afb_list_1_no_freq: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:NUMBer
driver.source.bb.radio.fm.rds.af.b.list1.set_number(afb_list_1_no_freq = 1)
```

Sets the number of frequencies of a list in AF method B.

param afb_list_1_no_freq
integer Range: 0 to 12

set_tfrequency(*af_list_1_tun_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:TFRequency
driver.source.bb.radio.fm.rds.af.b.list1.set_tfrequency(af_list_1_tun_freq = 1.
↪0)
```

Sets the tuning frequency of a list in AF method B.

param af_list_1_tun_freq
float Range: 87.6 to 107.9, Unit: MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list1.clone()
```

Subgroups

6.18.3.23.2.14 Desc<AlternaiveFreqList>

RepCap Settings

```
# Range: Nr1 .. Nr12
rc = driver.source.bb.radio.fm.rds.af.b.list1.desc.repcap_alternaiveFreqList_get()
driver.source.bb.radio.fm.rds.af.b.list1.desc.repcap_alternaiveFreqList_set(repcap.
↪AlternaiveFreqList.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:DESC<CH>
```

class DescCls

Desc commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: AlternaiveFreqList, default value after init: AlternaiveFreqList.Nr1

get(*alternaiveFreqList=AlternaiveFreqList.Default*) → TxAudioBcFmRdsAfBorder

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:DESC<CH>
value: enums.TxAudioBcFmRdsAfBorder = driver.source.bb.radio.fm.rds.af.b.list1.
↪desc.get(alternaiveFreqList = repcap.AlternaiveFreqList.Default)
```

Sets the frequency order of the corresponding number of the selected list.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

return
af_list_1_order: No help available

set(*af_list_1_order: TxAudioBcFmRdsAfBorder*, *alternaiveFreqList=AlternaiveFreqList.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:DESC<CH>
driver.source.bb.radio.fm.rds.af.b.list1.desc.set(af_list_1_order = enums.
↪TxAudioBcFmRdsAfBorder.ASC, alternaiveFreqList = repcap.AlternaiveFreqList.
↪Default)
```

Sets the frequency order of the corresponding number of the selected list.

param af_list_1_order
ASC| DESC ASC Ascending order, the same program is carried. DESC Descending order, the alternative frequency points to a program that has regional variants.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list1.desc.clone()
```

6.18.3.23.2.15 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.af.b.list1.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.af.b.list1.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.af.b.list1.frequency.get(index =␣
↪repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
af_list_1_freq: No help available

set(*af_list_1_freq: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST1:FREQuency<CH>
driver.source.bb.radio.fm.rds.af.b.list1.frequency.set(af_list_1_freq = 1.0,␣
↪index = repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param af_list_1_freq
float Range: 87.6 to 107.9, Unit: MHz

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list1.frequency.clone()
```

6.18.3.23.2.16 List2

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:NUMBer
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:TFRequency
```

class List2Cls

List2 commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:NUMBer
value: int = driver.source.bb.radio.fm.rds.af.b.list2.get_number()
```

Sets the number of frequencies of a list in AF method B.

return
afb_list_2_no_freq: No help available

get_tfrequency() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:TFRequency
value: float = driver.source.bb.radio.fm.rds.af.b.list2.get_tfrequency()
```

Sets the tuning frequency of a list in AF method B.

return
af_list_2_tun_freq: No help available

set_number(*afb_list_2_no_freq: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:NUMBer
driver.source.bb.radio.fm.rds.af.b.list2.set_number(afb_list_2_no_freq = 1)
```

Sets the number of frequencies of a list in AF method B.

param afb_list_2_no_freq
integer Range: 0 to 12

set_tfrequency(*af_list_2_tun_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:TFRequency
driver.source.bb.radio.fm.rds.af.b.list2.set_tfrequency(af_list_2_tun_freq = 1.
↪0)
```

Sets the tuning frequency of a list in AF method B.

param af_list_2_tun_freq
float Range: 87.6 to 107.9, Unit: MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list2.clone()
```

Subgroups

6.18.3.23.2.17 Desc<AlternaiveFreqList>

RepCap Settings

```
# Range: Nr1 .. Nr12
rc = driver.source.bb.radio.fm.rds.af.b.list2.desc.repcap_alternaiveFreqList_get()
driver.source.bb.radio.fm.rds.af.b.list2.desc.repcap_alternaiveFreqList_set(repcap.
↪AlternaiveFreqList.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:DESC<CH>
```

class DescCls

Desc commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: AlternaiveFreqList, default value after init: AlternaiveFreqList.Nr1

get(*alternaiveFreqList=AlternaiveFreqList.Default*) → TxAudioBcFmRdsAfBorder

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:DESC<CH>
value: enums.TxAudioBcFmRdsAfBorder = driver.source.bb.radio.fm.rds.af.b.list2.
↪desc.get(alternaiveFreqList = repcap.AlternaiveFreqList.Default)
```

Sets the frequency order of the corresponding number of the selected list.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

return
af_list_2_order: No help available

set(*af_list_2_order: TxAudioBcFmRdsAfBorder*, *alternaiveFreqList=AlternaiveFreqList.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:DESC<CH>
driver.source.bb.radio.fm.rds.af.b.list2.desc.set(af_list_2_order = enums.
↪TxAudioBcFmRdsAfBorder.ASC, alternaiveFreqList = repcap.AlternaiveFreqList.
↪Default)
```

Sets the frequency order of the corresponding number of the selected list.

param af_list_2_order
ASC| DESC ASC Ascending order, the same program is carried. DESC Descending order, the alternative frequency points to a program that has regional variants.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list2.desc.clone()
```

6.18.3.23.2.18 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.af.b.list2.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.af.b.list2.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.af.b.list2.frequency.get(index =␣
↪repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
af_list_2_freq: No help available

set(*af_list_2_freq: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST2:FREQuency<CH>
driver.source.bb.radio.fm.rds.af.b.list2.frequency.set(af_list_2_freq = 1.0,␣
↪index = repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param af_list_2_freq
float Range: 87.6 to 107.9, Unit: MHz

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list2.frequency.clone()
```

6.18.3.23.2.19 List3

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:NUMBer
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:TFRequency
```

class List3Cls

List3 commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:NUMBer
value: int = driver.source.bb.radio.fm.rds.af.b.list3.get_number()
```

Sets the number of frequencies of a list in AF method B.

return
afb_list_3_no_freq: No help available

get_tfrequency() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:TFRequency
value: float = driver.source.bb.radio.fm.rds.af.b.list3.get_tfrequency()
```

Sets the tuning frequency of a list in AF method B.

return
af_list_3_tun_freq: No help available

set_number(*afb_list_3_no_freq: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:NUMBer
driver.source.bb.radio.fm.rds.af.b.list3.set_number(afb_list_3_no_freq = 1)
```

Sets the number of frequencies of a list in AF method B.

param afb_list_3_no_freq
integer Range: 0 to 12

set_tfrequency(*af_list_3_tun_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:TFRequency
driver.source.bb.radio.fm.rds.af.b.list3.set_tfrequency(af_list_3_tun_freq = 1.
↪0)
```

Sets the tuning frequency of a list in AF method B.

param af_list_3_tun_freq
float Range: 87.6 to 107.9, Unit: MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list3.clone()
```

Subgroups

6.18.3.23.2.20 Desc<AlternaiveFreqList>

RepCap Settings

```
# Range: Nr1 .. Nr12
rc = driver.source.bb.radio.fm.rds.af.b.list3.desc.repcap_alternaiveFreqList_get()
driver.source.bb.radio.fm.rds.af.b.list3.desc.repcap_alternaiveFreqList_set(repcap.
↪AlternaiveFreqList.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:DESC<CH>
```

class DescCls

Desc commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: AlternaiveFreqList, default value after init: AlternaiveFreqList.Nr1

get(*alternaiveFreqList=AlternaiveFreqList.Default*) → TxAudioBcFmRdsAfBorder

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:DESC<CH>
value: enums.TxAudioBcFmRdsAfBorder = driver.source.bb.radio.fm.rds.af.b.list3.
↪desc.get(alternaiveFreqList = repcap.AlternaiveFreqList.Default)
```

Sets the frequency order of the corresponding number of the selected list.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

return
af_list_3_order: No help available

set(*af_list_3_order: TxAudioBcFmRdsAfBorder*, *alternaiveFreqList=AlternaiveFreqList.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:DESC<CH>
driver.source.bb.radio.fm.rds.af.b.list3.desc.set(af_list_3_order = enums.
↪TxAudioBcFmRdsAfBorder.ASC, alternaiveFreqList = repcap.AlternaiveFreqList.
↪Default)
```

Sets the frequency order of the corresponding number of the selected list.

param af_list_3_order
ASC| DESC ASC Ascending order, the same program is carried. DESC Descending order, the alternative frequency points to a program that has regional variants.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list3.desc.clone()
```

6.18.3.23.2.21 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.af.b.list3.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.af.b.list3.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.af.b.list3.frequency.get(index =␣
↪repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
af_list_3_freq: No help available

set(*af_list_3_freq: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST3:FREQuency<CH>
driver.source.bb.radio.fm.rds.af.b.list3.frequency.set(af_list_3_freq = 1.0,␣
↪index = repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param af_list_3_freq
float Range: 87.6 to 107.9, Unit: MHz

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list3.frequency.clone()
```

6.18.3.23.2.22 List4

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:NUMBer
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:TFRequency
```

class List4Cls

List4 commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:NUMBer
value: int = driver.source.bb.radio.fm.rds.af.b.list4.get_number()
```

Sets the number of frequencies of a list in AF method B.

return
afb_list_4_no_freq: No help available

get_tfrequency() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:TFRequency
value: float = driver.source.bb.radio.fm.rds.af.b.list4.get_tfrequency()
```

Sets the tuning frequency of a list in AF method B.

return
af_list_4_tun_freq: No help available

set_number(*afb_list_4_no_freq: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:NUMBer
driver.source.bb.radio.fm.rds.af.b.list4.set_number(afb_list_4_no_freq = 1)
```

Sets the number of frequencies of a list in AF method B.

param afb_list_4_no_freq
integer Range: 0 to 12

set_tfrequency(*af_list_4_tun_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:TFRequency
driver.source.bb.radio.fm.rds.af.b.list4.set_tfrequency(af_list_4_tun_freq = 1.
↪0)
```

Sets the tuning frequency of a list in AF method B.

param af_list_4_tun_freq
float Range: 87.6 to 107.9, Unit: MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list4.clone()
```

Subgroups

6.18.3.23.2.23 Desc<AlternaiveFreqList>

RepCap Settings

```
# Range: Nr1 .. Nr12
rc = driver.source.bb.radio.fm.rds.af.b.list4.desc.repcap_alternaiveFreqList_get()
driver.source.bb.radio.fm.rds.af.b.list4.desc.repcap_alternaiveFreqList_set(repcap.
↪AlternaiveFreqList.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:DESC<CH>
```

class DescCls

Desc commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: AlternaiveFreqList, default value after init: AlternaiveFreqList.Nr1

get(*alternaiveFreqList=AlternaiveFreqList.Default*) → TxAudioBcFmRdsAfBorder

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:DESC<CH>
value: enums.TxAudioBcFmRdsAfBorder = driver.source.bb.radio.fm.rds.af.b.list4.
↪desc.get(alternaiveFreqList = repcap.AlternaiveFreqList.Default)
```

Sets the frequency order of the corresponding number of the selected list.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

return
af_list_4_order: No help available

set(*af_list_4_order: TxAudioBcFmRdsAfBorder*, *alternaiveFreqList=AlternaiveFreqList.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:DESC<CH>
driver.source.bb.radio.fm.rds.af.b.list4.desc.set(af_list_4_order = enums.
↪TxAudioBcFmRdsAfBorder.ASC, alternaiveFreqList = repcap.AlternaiveFreqList.
↪Default)
```

Sets the frequency order of the corresponding number of the selected list.

param af_list_4_order
ASC| DESC ASC Ascending order, the same program is carried. DESC Descending order, the alternative frequency points to a program that has regional variants.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list4.desc.clone()
```

6.18.3.23.2.24 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.af.b.list4.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.af.b.list4.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.af.b.list4.frequency.get(index =␣
↪repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
af_list_4_freq: No help available

set(*af_list_4_freq: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST4:FREQuency<CH>
driver.source.bb.radio.fm.rds.af.b.list4.frequency.set(af_list_4_freq = 1.0,␣
↪index = repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param af_list_4_freq
float Range: 87.6 to 107.9, Unit: MHz

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list4.frequency.clone()
```

6.18.3.23.2.25 List5

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:NUMBer
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:TFRequency
```

class List5Cls

List5 commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:NUMBer
value: int = driver.source.bb.radio.fm.rds.af.b.list5.get_number()
```

Sets the number of frequencies of a list in AF method B.

return
afb_list_5_no_freq: integer Range: 0 to 12

get_tfrequency() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:TFRequency
value: float = driver.source.bb.radio.fm.rds.af.b.list5.get_tfrequency()
```

Sets the tuning frequency of a list in AF method B.

return
af_list_5_tun_freq: float Range: 87.6 to 107.9, Unit: MHz

set_number(*afb_list_5_no_freq: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:NUMBer
driver.source.bb.radio.fm.rds.af.b.list5.set_number(afb_list_5_no_freq = 1)
```

Sets the number of frequencies of a list in AF method B.

param afb_list_5_no_freq
integer Range: 0 to 12

set_tfrequency(*af_list_5_tun_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:TFRequency
driver.source.bb.radio.fm.rds.af.b.list5.set_tfrequency(af_list_5_tun_freq = 1.
↪0)
```

Sets the tuning frequency of a list in AF method B.

param af_list_5_tun_freq
float Range: 87.6 to 107.9, Unit: MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list5.clone()
```

Subgroups

6.18.3.23.2.26 Desc<AlternaiveFreqList>

RepCap Settings

```
# Range: Nr1 .. Nr12
rc = driver.source.bb.radio.fm.rds.af.b.list5.desc.repcap_alternaiveFreqList_get()
driver.source.bb.radio.fm.rds.af.b.list5.desc.repcap_alternaiveFreqList_set(repcap.
↪AlternaiveFreqList.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:DESC<CH>
```

class DescCls

Desc commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: AlternaiveFreqList, default value after init: AlternaiveFreqList.Nr1

get(*alternaiveFreqList=AlternaiveFreqList.Default*) → TxAudioBcFmRdsAfBorder

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:DESC<CH>
value: enums.TxAudioBcFmRdsAfBorder = driver.source.bb.radio.fm.rds.af.b.list5.
↪desc.get(alternaiveFreqList = repcap.AlternaiveFreqList.Default)
```

Sets the frequency order of the corresponding number of the selected list.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

return
af_list_5_order: ASC| DESC ASC Ascending order, the same program is carried. DESC Descending order, the alternative frequency points to a program that has regional variants.

set(*af_list_5_order: TxAudioBcFmRdsAfBorder*, *alternaiveFreqList=AlternaiveFreqList.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:DESC<CH>
driver.source.bb.radio.fm.rds.af.b.list5.desc.set(af_list_5_order = enums.
↪TxAudioBcFmRdsAfBorder.ASC, alternaiveFreqList = repcap.AlternaiveFreqList.
↪Default)
```

Sets the frequency order of the corresponding number of the selected list.

param af_list_5_order
ASC| DESC ASC Ascending order, the same program is carried. DESC Descending order, the alternative frequency points to a program that has regional variants.

param alternaiveFreqList
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Desc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list5.desc.clone()
```

6.18.3.23.2.27 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.af.b.list5.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.af.b.list5.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.af.b.list5.frequency.get(index =␣
↪repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
af_list_5_freq: float Range: 87.6 to 107.9, Unit: MHz

set(*af_list_5_freq: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:AF:B:LIST5:FREQuency<CH>
driver.source.bb.radio.fm.rds.af.b.list5.frequency.set(af_list_5_freq = 1.0,␣
↪index = repcap.Index.Default)
```

Sets an alternative frequency of a list in AF method B.

param af_list_5_freq
float Range: 87.6 to 107.9, Unit: MHz

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.af.b.list5.frequency.clone()
```

6.18.3.23.2.28 Di

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:DI:ARTificial
[SOURce<HW>]:BB:RADio:FM:RDS:DI:COMPressed
[SOURce<HW>]:BB:RADio:FM:RDS:DI:DYNamic
[SOURce<HW>]:BB:RADio:FM:RDS:DI:STEReo
```

class DiCls

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

get_artificial() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:ARTificial
value: bool = driver.source.bb.radio.fm.rds.di.get_artificial()
```

Enables/disables 'artificial head' decoder identification.

return
di_artifical: 1| ON| 0| OFF

get_compressed() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:COMPressed
value: bool = driver.source.bb.radio.fm.rds.di.get_compressed()
```

Enables/disables compressed decoder identification.

return
di_compressed: 1| ON| 0| OFF

get_dynamic() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:DYNamic
value: bool = driver.source.bb.radio.fm.rds.di.get_dynamic()
```

Enables/disables dynamic decoder identification.

return
di_dynamic: 1| ON| 0| OFF

get_stereo() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:STEReo
value: bool = driver.source.bb.radio.fm.rds.di.get_stereo()
```

Enables/disables stereo decoder identification.

return
di_stereo: 1| ON| 0| OFF ON Stereo transmission OFF Mono transmission

set_artificial(*di_artifical: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:ARTificial
driver.source.bb.radio.fm.rds.di.set_artificial(di_artifical = False)
```

Enables/disables 'artificial head' decoder identification.

param di_artifical
1| ON| 0| OFF

set_compressed(*di_compressed: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:COMPressed
driver.source.bb.radio.fm.rds.di.set_compressed(di_compressed = False)
```

Enables/disables compressed decoder identification.

param di_compressed
1| ON| 0| OFF

set_dynamic(*di_dynamic: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:DYNamic
driver.source.bb.radio.fm.rds.di.set_dynamic(di_dynamic = False)
```

Enables/disables dynamic decoder identification.

param di_dynamic
1| ON| 0| OFF

set_stereo(*di_stereo: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:DI:STEReo
driver.source.bb.radio.fm.rds.di.set_stereo(di_stereo = False)
```

Enables/disables stereo decoder identification.

param di_stereo
1| ON| 0| OFF ON Stereo transmission OFF Mono transmission

6.18.3.23.2.29 Eon

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:EON:EG
[SOURce<HW>]:BB:RADio:FM:RDS:EON:ILS
[SOURce<HW>]:BB:RADio:FM:RDS:EON:LA
[SOURce<HW>]:BB:RADio:FM:RDS:EON:LSN
[SOURce<HW>]:BB:RADio:FM:RDS:EON:PI
[SOURce<HW>]:BB:RADio:FM:RDS:EON:PIN
[SOURce<HW>]:BB:RADio:FM:RDS:EON:PS
[SOURce<HW>]:BB:RADio:FM:RDS:EON:PTY
[SOURce<HW>]:BB:RADio:FM:RDS:EON:TP
[SOURce<HW>]:BB:RADio:FM:RDS:EON:Ta
```

class EonCls

Eon commands group definition. 16 total commands, 1 Subgroups, 10 group commands

get_eg() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:EG
value: bool = driver.source.bb.radio.fm.rds.eon.get_eg()
```

Enables the enhanced other network extended generic indicator.

return
eon_eg: 1| ON| 0| OFF

get_ils() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:ILS
value: bool = driver.source.bb.radio.fm.rds.eon.get_ils()
```

Enables the enhanced other network international linkage set indicator.

return
eon_ils: 1| ON| 0| OFF

get_la() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:LA
value: bool = driver.source.bb.radio.fm.rds.eon.get_la()
```

Enables the enhanced other network linkage actuator.

return
eon_la: 1| ON| 0| OFF

get_lsn() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:LSN
value: int = driver.source.bb.radio.fm.rds.eon.get_lsn()
```

Sets the enhanced other network linkage set number. The LSN comprises a 12-bit value in decimal representation.

return
eon_lsn: integer Range: 0 to 4095

get_pi() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PI
value: int = driver.source.bb.radio.fm.rds.eon.get_pi()
```

Sets the enhanced other network program identification.

return
eon_pi: integer Range: 0 to 65535

get_pin() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PIN
value: int = driver.source.bb.radio.fm.rds.eon.get_pin()
```

Sets the enhanced other network program item number.

return
eon_pin: integer Range: 0 to 65535

get_ps() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PS
value: str = driver.source.bb.radio.fm.rds.eon.get_ps()
```

Sets the enhanced other network program service name.

return
eon_ps: string

get_pty() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PTY
value: int = driver.source.bb.radio.fm.rds.eon.get_pty()
```

Sets the enhanced other network program type.

return
eon_pty: integer Range: 0 to 31

get_ta() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:Ta
value: bool = driver.source.bb.radio.fm.rds.eon.get_ta()
```

Enables the enhanced other network traffic announcement.

return
eon_ta: 1| ON| 0| OFF

get_tp() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:TP
value: bool = driver.source.bb.radio.fm.rds.eon.get_tp()
```

Enables the enhanced other network traffic program.

return
eon_tp: 1| ON| 0| OFF

set_eg(*eon_eg: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:EG
driver.source.bb.radio.fm.rds.eon.set_eg(eon_eg = False)
```

Enables the enhanced other network extended generic indicator.

param eon_eg
1| ON| 0| OFF

set_ils(*eon_ils: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:ILS
driver.source.bb.radio.fm.rds.eon.set_ils(eon_ils = False)
```

Enables the enhanced other network international linkage set indicator.

param eon_ils
1| ON| 0| OFF

set_la(*eon_la: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:LA
driver.source.bb.radio.fm.rds.eon.set_la(eon_la = False)
```

Enables the enhanced other network linkage actuator.

param eon_la
1| ON| 0| OFF

set_lsn(*eon_lsn: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:LSN
driver.source.bb.radio.fm.rds.eon.set_lsn(eon_lsn = 1)
```

Sets the enhanced other network linkage set number. The LSN comprises a 12-bit value in decimal representation.

param eon_lsn
integer Range: 0 to 4095

set_pi(*eon_pi: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PI
driver.source.bb.radio.fm.rds.eon.set_pi(eon_pi = 1)
```

Sets the enhanced other network program identification.

param eon_pi
integer Range: 0 to 65535

set_pin(*eon_pin: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PIN
driver.source.bb.radio.fm.rds.eon.set_pin(eon_pin = 1)
```

Sets the enhanced other network program item number.

param eon_pin
integer Range: 0 to 65535

set_ps(*eon_ps: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PS
driver.source.bb.radio.fm.rds.eon.set_ps(eon_ps = 'abc')
```

Sets the enhanced other network program service name.

param eon_ps
string

set_pty(*eon_pty: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:PTY
driver.source.bb.radio.fm.rds.eon.set_pty(eon_pty = 1)
```

Sets the enhanced other network program type.

param eon_pty
integer Range: 0 to 31

set_ta(*eon_ta: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:Ta
driver.source.bb.radio.fm.rds.eon.set_ta(eon_ta = False)
```

Enables the enhanced other network traffic announcement.

param eon_ta
1| ON| 0| OFF

set_tp(*eon_tp: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:TP
driver.source.bb.radio.fm.rds.eon.set_tp(eon_tp = False)
```

Enables the enhanced other network traffic program.

param eon_tp
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.eon.clone()
```

Subgroups

6.18.3.23.2.30 Af

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:METHod
```

class AfCls

Af commands group definition. 6 total commands, 2 Subgroups, 1 group commands

get_method() → TxAudioBcFmRdsEonAfMethod

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:METHod
value: enums.TxAudioBcFmRdsEonAfMethod = driver.source.bb.radio.fm.rds.eon.af.
↪get_method()
```

No command help available

return
eon_af_method: MAPF| A

set_method(*eon_af_method: TxAudioBcFmRdsEonAfMethod*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:METHod
driver.source.bb.radio.fm.rds.eon.af.set_method(eon_af_method = enums.
↪TxAudioBcFmRdsEonAfMethod.A)
```

No command help available

param eon_af_method
MAPF| A

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.eon.af.clone()
```

Subgroups

6.18.3.23.2.31 A

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:A:NUMBer
```

class ACls

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

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:A:NUMBer
value: int = driver.source.bb.radio.fm.rds.eon.af.a.get_number()
```

Defines the number of alternative frequencies.

return
eon_afa_num_freq: integer Range: 0 to 25

set_number(*eon_afa_num_freq: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:A:NUMBer
driver.source.bb.radio.fm.rds.eon.af.a.set_number(eon_afa_num_freq = 1)
```

Defines the number of alternative frequencies.

param eon_afa_num_freq
integer Range: 0 to 25

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.eon.af.a.clone()
```

Subgroups

6.18.3.23.2.32 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.eon.af.a.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.eon.af.a.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:A:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:A:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.eon.af.a.frequency.get(index =␣
↪repcap.Index.Default)
```

Sets the alternative frequencies in EON, AF method A.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
eon_afa_freq: float Range: 87.6 to 107.9, Unit: MHz

set(*eon_afa_freq: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:A:FREQuency<CH>
driver.source.bb.radio.fm.rds.eon.af.a.frequency.set(eon_afa_freq = 1.0, index␣
↪= repcap.Index.Default)
```

Sets the alternative frequencies in EON, AF method A.

param eon_afa_freq
float Range: 87.6 to 107.9, Unit: MHz

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.eon.af.a.frequency.clone()
```

6.18.3.23.2.33 B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:NUMBer
[SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:TFRequency
```

class BCls

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

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:NUMBer
value: int = driver.source.bb.radio.fm.rds.eon.af.b.get_number()
```

No command help available

return
eon_afb_num_freq: integer Range: 0 to 4

get_tfrequency() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:TFRequency
value: float = driver.source.bb.radio.fm.rds.eon.af.b.get_tfrequency()
```

Sets the tuning frequency of in AF mapped frequencies method.

return
eon_aft_freq: float Range: 87.6 to 107.9, Unit: MHz

set_number(*eon_afb_num_freq: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:NUMBer
driver.source.bb.radio.fm.rds.eon.af.b.set_number(eon_afb_num_freq = 1)
```

No command help available

param eon_afb_num_freq
integer Range: 0 to 4

set_tfrequency(*eon_aft_freq: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:TFRequency
driver.source.bb.radio.fm.rds.eon.af.b.set_tfrequency(eon_aft_freq = 1.0)
```

Sets the tuning frequency of in AF mapped frequencies method.

param eon_aft_freq
float Range: 87.6 to 107.9, Unit: MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.eon.af.b.clone()
```

Subgroups

6.18.3.23.2.34 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.radio.fm.rds.eon.af.b.frequency.repcap_index_get()
driver.source.bb.radio.fm.rds.eon.af.b.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:FREQuency<CH>
value: float = driver.source.bb.radio.fm.rds.eon.af.b.frequency.get(index =␣
↪repcap.Index.Default)
```

Sets the alternative frequencies in EON for AF mapped frequencies method.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
eon_afb_freq: float Range: 87.6 to 107.9, Unit: MHz

set(*eon_afb_freq: float*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:EON:AF:B:FREQuency<CH>
driver.source.bb.radio.fm.rds.eon.af.b.frequency.set(eon_afb_freq = 1.0, index
↪= repcap.Index.Default)
```

Sets the alternative frequencies in EON for AF mapped frequencies method.

param eon_afb_freq
float Range: 87.6 to 107.9, Unit: MHz

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.eon.af.b.frequency.clone()
```

6.18.3.23.2.35 Group

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:GROup:SEQuence
```

class GroupCls

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

get_sequence() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:GROup:SEQuence
value: str = driver.source.bb.radio.fm.rds.group.get_sequence()
```

No command help available

return
group_sequence: string

set_sequence(*group_sequence: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:GROup:SEQuence
driver.source.bb.radio.fm.rds.group.set_sequence(group_sequence = 'abc')
```

No command help available

param group_sequence
string

6.18.3.23.2.36 Opf

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:[STATe]
```

class OpfCls

Opf commands group definition. 71 total commands, 24 Subgroups, 1 group commands

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:[STATe]
value: bool = driver.source.bb.radio.fm.rds.opf.get_state()
```

Enables the open format.

return
open_format_state: 1| ON| 0| OFF

set_state(*open_format_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:[STATe]
driver.source.bb.radio.fm.rds.opf.set_state(open_format_state = False)
```

Enables the open format.

param open_format_state
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.opf.clone()
```

Subgroups

6.18.3.23.2.37 Apply

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:APPLy
```

class ApplyCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:APPLy
driver.source.bb.radio.fm.rds.opf.apply.set()
```

Triggers transmission of open format data.

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

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:APPLy
driver.source.bb.radio.fm.rds.opf.apply.set_with_opc()
```

Triggers transmission of open format data.

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

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

6.18.3.23.2.38 G10B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk4
```

class G10BCls

G10B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g10B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g10B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g10B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g10B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G10B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g10B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.39 G11A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk4
```

class G11ACls

G11A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g11A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g11A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g11A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g11A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g11A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g11A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.40 G11B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk4
```

class G11BCls

G11B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g11B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g11B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g11B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g11B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G11B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g11B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.41 G12A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk4
```

class G12ACls

G12A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g12A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g12A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g12A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g12A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g12A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g12A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.42 G12B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk4
```

class G12BCls

G12B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g12B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g12B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g12B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g12B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G12B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g12B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.43 G13A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk4
```

class G13ACls

G13A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g13A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g13A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g13A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g13A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g13A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g13A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.44 G13B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk4
```

class G13BCls

G13B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g13B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g13B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g13B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g13B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G13B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g13B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.45 G15A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk4
```

class G15ACls

G15A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g15A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g15A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g15A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g15A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g15A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G15A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g15A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.46 G1A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk4
```

class G1ACls

G1A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g1A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g1A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g1A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g1A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g1A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g1A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.47 G1B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk4
```

class G1BCls

G1B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g1B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g1B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g1B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g1B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G1B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g1B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.48 G3A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk4
```

class G3ACls

G3A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g3A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g3A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g3A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g3A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g3A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g3A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.49 G3B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk4
```

class G3BCls

G3B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g3B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g3B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g3B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g3B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G3B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g3B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.50 G4B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk4
```

class G4BCls

G4B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g4B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g4B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g4B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g4B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G4B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g4B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.51 G5A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk4
```

class G5ACls

G5A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g5A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g5A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g5A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g5A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g5A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g5A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.52 G5B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk4
```

class G5BCls

G5B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g5B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g5B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g5B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g5B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G5B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g5B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.53 G6A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk4
```

class G6ACls

G6A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g6A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g6A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g6A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g6A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g6A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g6A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.54 G6B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk4
```

class G6BCls

G6B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g6B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g6B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g6B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g6B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G6B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g6B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.55 G7A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk4
```

class G7ACls

G7A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g7A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g7A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g7A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g7A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g7A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g7A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.56 G7B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk4
```

class G7BCls

G7B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g7B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g7B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g7B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g7B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G7B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g7B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.57 G8A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk4
```

class G8ACls

G8A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g8A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g8A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g8A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g8A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g8A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g8A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.58 G8B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk4
```

class G8BCls

G8B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g8B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g8B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g8B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g8B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G8B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g8B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.59 G9A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk4
```

class G9ACls

G9A commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g9A.get_block_2()
```

Sets block 4 of the open format group types A.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g9A.get_block_3()
```

Sets block 4 of the open format group types A.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g9A.get_block_4()
```

Sets block 4 of the open format group types A.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk2
driver.source.bb.radio.fm.rds.opf.g9A.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types A.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_3(*open_format_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk3
driver.source.bb.radio.fm.rds.opf.g9A.set_block_3(open_format_blk_3 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_3
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9A:BLOCk4
driver.source.bb.radio.fm.rds.opf.g9A.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types A.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.60 G9B

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk3
[SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk4
```

class G9BCls

G9B commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk2
value: int = driver.source.bb.radio.fm.rds.opf.g9B.get_block_2()
```

Sets block 4 of the open format group types B.

return
ofg_1_ablk_2: No help available

get_block_3() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk3
value: int = driver.source.bb.radio.fm.rds.opf.g9B.get_block_3()
```

Sets block 4 of the open format group types B.

return
open_format_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk4
value: int = driver.source.bb.radio.fm.rds.opf.g9B.get_block_4()
```

Sets block 4 of the open format group types B.

return
open_format_blk_4: integer Range: 0 to 65535

set_block_2(*ofg_1_ablk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk2
driver.source.bb.radio.fm.rds.opf.g9B.set_block_2(ofg_1_ablk_2 = 1)
```

Sets block 4 of the open format group types B.

param ofg_1_ablk_2
integer Range: 0 to 65535

set_block_4(*open_format_blk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:OPF:G9B:BLOCk4
driver.source.bb.radio.fm.rds.opf.g9B.set_block_4(open_format_blk_4 = 1)
```

Sets block 4 of the open format group types B.

param open_format_blk_4
integer Range: 0 to 65535

6.18.3.23.2.61 Tmc

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:READy
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:[STATe]
```

class TmcCls

Tmc commands group definition. 8 total commands, 3 Subgroups, 2 group commands

get_ready() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:READy
value: bool = driver.source.bb.radio.fm.rds.tmc.get_ready()
```

No command help available

return
tmc_ready: 1| ON| 0| OFF

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:[STATe]
value: bool = driver.source.bb.radio.fm.rds.tmc.get_state()
```

Enables the traffic message channel.

return
tmc_state: 1| ON| 0| OFF

set_state(*tmc_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:[STATe]
driver.source.bb.radio.fm.rds.tmc.set_state(tmc_state = False)
```

Enables the traffic message channel.

param tmc_state
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.tmc.clone()
```

Subgroups

6.18.3.23.2.62 Apply

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:APPLy
```

class ApplyCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:APPLy
driver.source.bb.radio.fm.rds.tmc.apply.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:APPLy
driver.source.bb.radio.fm.rds.tmc.apply.set_with_opc()
```

No command help available

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

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

6.18.3.23.2.63 G3A

class G3ACls

G3A 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.bb.radio.fm.rds.tmc.g3A.clone()
```

Subgroups

6.18.3.23.2.64 Var<GroupTypeVariant>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.source.bb.radio.fm.rds.tmc.g3A.var.repcap_groupTypeVariant_get()
driver.source.bb.radio.fm.rds.tmc.g3A.var.repcap_groupTypeVariant_set(repcap.
↪GroupTypeVariant.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:G3A:VAR<CH>
```

class VarCls

Var commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: GroupTypeVariant, default value after init: GroupTypeVariant.Nr1

get(*groupTypeVariant=GroupTypeVariant.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G3A:VAR<CH>
value: int = driver.source.bb.radio.fm.rds.tmc.g3A.var.get(groupTypeVariant =␣
↪repcap.GroupTypeVariant.Default)
```

Sets the traffic message channel 3A group variants.

param groupTypeVariant
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Var')

return
g_3_avar: integer Range: 0 to 65535

set(*g_3_avar: int*, *groupTypeVariant=GroupTypeVariant.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G3A:VAR<CH>
driver.source.bb.radio.fm.rds.tmc.g3A.var.set(g_3_avar = 1, groupTypeVariant =␣
↪repcap.GroupTypeVariant.Default)
```

Sets the traffic message channel 3A group variants.

param g_3_avar
integer Range: 0 to 65535

param groupTypeVariant
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Var')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.rds.tmc.g3A.var.clone()
```

6.18.3.23.2.65 G8A

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk2
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk3a
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk4
[SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:NUMBer
```

class G8ACls

G8A commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_block_2() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk2
value: int = driver.source.bb.radio.fm.rds.tmc.g8A.get_block_2()
```

No command help available

return
g_8_ab_lk_2: No help available

get_block_3_a() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk3a
value: int = driver.source.bb.radio.fm.rds.tmc.g8A.get_block_3_a()
```

No command help available

return
ga_8_blk_3: No help available

get_block_4() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk4
value: int = driver.source.bb.radio.fm.rds.tmc.g8A.get_block_4()
```

No command help available

return
g_8_ab_lk_4: integer Range: 0 to 65535

get_number() → int

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:NUMBer
value: int = driver.source.bb.radio.fm.rds.tmc.g8A.get_number()
```

Defines the number of A8 groups.

return
g_8_ano: integer Range: 1 to 6

set_block_2(*g_8_ab_lk_2: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk2
driver.source.bb.radio.fm.rds.tmc.g8A.set_block_2(g_8_ab_lk_2 = 1)
```

No command help available

param g_8_ab_lk_2
integer Range: 0 to 65535

set_block_3_a(*ga_8_blk_3: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk3a
driver.source.bb.radio.fm.rds.tmc.g8A.set_block_3_a(ga_8_blk_3 = 1)
```

No command help available

param ga_8_blk_3
integer Range: 0 to 65535

set_block_4(*g_8_ab_lk_4: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:BLOCk4
driver.source.bb.radio.fm.rds.tmc.g8A.set_block_4(g_8_ab_lk_4 = 1)
```

No command help available

param g_8_ab_lk_4
integer Range: 0 to 65535

set_number(*g_8_ano: int*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TMC:G8A:NUMBer
driver.source.bb.radio.fm.rds.tmc.g8A.set_number(g_8_ano = 1)
```

Defines the number of A8 groups.

param g_8_ano
integer Range: 1 to 6

6.18.3.23.2.66 Tp

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:RDS:TP:[STATe]
```

class TpCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TP:[STATe]
value: bool = driver.source.bb.radio.fm.rds.tp.get_state()
```

Enable/disables the traffic program flag.

return
tp: 1| ON| 0| OFF

set_state(*tp: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:RDS:TP:[STATe]
driver.source.bb.radio.fm.rds.tp.set_state(tp = False)
```

Enable/disables the traffic program flag.

param tp
1| ON| 0| OFF

6.18.3.23.2.67 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:SETTing:CATalog
[SOURce<HW>]:BB:RADio:FM:SETTing:DELete
[SOURce<HW>]:BB:RADio:FM:SETTing:LOAD
[SOURce<HW>]:BB:RADio:FM:SETTing:STORe
```

class SettingCls

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

delete(*fm_del: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:SETTing:DELete
driver.source.bb.radio.fm.setting.delete(fm_del = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param fm_del
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:SETTing:CATalog
value: List[str] = driver.source.bb.radio.fm.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
tx_audio_bc_fm_cat_name: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:SETTing:LOAD
value: str = driver.source.bb.radio.fm.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
fm_rcl: 'filename' Filename or complete file path; file extension can be omitted

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:SETTing:STORe
value: str = driver.source.bb.radio.fm.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.am/fm/rds) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
fm_sav: 'filename' Filename or complete file path

set_load(*fm_rcl: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:SETTing:LOAD
driver.source.bb.radio.fm.setting.set_load(fm_rcl = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.am/fm/rds. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param fm_rcl
'filename' Filename or complete file path; file extension can be omitted

set_store(*fm_sav: str*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:SETTing:STORe
driver.source.bb.radio.fm.setting.set_store(fm_sav = 'abc')
```

Saves the current settings into the selected file; the file extension (*.am/fm/rds) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param fm_sav
'filename' Filename or complete file path

6.18.3.23.2.68 Special

class SpecialCls

Special commands group definition. 4 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.radio.fm.special.clone()
```

Subgroups

6.18.3.23.2.69 Pilot

SCPI Commands :

```
[SOURce<HW>]:BB:RADio:FM:[SPECial]:PILot:PHASe
[SOURce<HW>]:BB:RADio:FM:[SPECial]:PILot:[STATe]
```

class PilotCls

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

get_phase() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:PILot:PHASe
value: float = driver.source.bb.radio.fm.special.pilot.get_phase()
```

Sets the phase offset of the 19 kHz pilot tone.

return
offset_pilot: float Range: -180 to 180

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:PILot:[STATe]
value: bool = driver.source.bb.radio.fm.special.pilot.get_state()
```

Enables/disables the 19 kHz pilot tone.

return
pilot: 1| ON| 0| OFF

set_phase(*offset_pilot: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:PILot:PHASe
driver.source.bb.radio.fm.special.pilot.set_phase(offset_pilot = 1.0)
```

Sets the phase offset of the 19 kHz pilot tone.

param offset_pilot
float Range: -180 to 180

set_state(*pilot: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:PILot:[STATe]
driver.source.bb.radio.fm.special.pilot.set_state(pilot = False)
```

Enables/disables the 19 kHz pilot tone.

param pilot
1| ON| 0| OFF

6.18.3.23.2.70 Rds

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:[SPECial]:RDS:PHASe
```

class RdsCls

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

get_phase() → float

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:RDS:PHASe
value: float = driver.source.bb.radio.fm.special.rds.get_phase()
```

Sets the phase offset of the suppressed 57 kHz carrier.

return
offset_rds: float Range: -180 to 180

set_phase(*offset_rds: float*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:RDS:PHASe
driver.source.bb.radio.fm.special.rds.set_phase(offset_rds = 1.0)
```

Sets the phase offset of the suppressed 57 kHz carrier.

param offset_rds
float Range: -180 to 180

6.18.3.23.2.71 Settings

SCPI Command :

```
[SOURce<HW>]:BB:RADio:FM:[SPECial]:SETTings:[STATe]
```

class SettingsCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:SETTings:[STATe]
value: bool = driver.source.bb.radio.fm.special.settings.get_state()
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

return
special_settings: 1| ON| 0| OFF

set_state(*special_settings: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:RADio:FM:[SPECial]:SETTings:[STATe]
driver.source.bb.radio.fm.special.settings.set_state(special_settings = False)
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

param special_settings
1| ON| 0| OFF

6.18.3.24 T2Dvb

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:LDATa
[SOURce<HW>]:BB:T2DVb:LF
[SOURce<HW>]:BB:T2DVb:NAUX
[SOURce<HW>]:BB:T2DVb:NETWorkmode
[SOURce<HW>]:BB:T2DVb:NSUB
[SOURce<HW>]:BB:T2DVb:NT2Frames
[SOURce<HW>]:BB:T2DVb:PAPR
[SOURce<HW>]:BB:T2DVb:PAYLoad
[SOURce<HW>]:BB:T2DVb:PID
[SOURce<HW>]:BB:T2DVb:PIDTestpack
[SOURce<HW>]:BB:T2DVb:PILot
[SOURce<HW>]:BB:T2DVb:PRESet
[SOURce<HW>]:BB:T2DVb:PROFile
[SOURce<HW>]:BB:T2DVb:SOURce
[SOURce<HW>]:BB:T2DVb:STATe
[SOURce<HW>]:BB:T2DVb:TFS
[SOURce<HW>]:BB:T2DVb:TSPacket
[SOURce<HW>]:BB:T2DVb:TXSYs
```

class T2DvbCls

T2Dvb commands group definition. 117 total commands, 15 Subgroups, 18 group commands

get_ldata() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:LDATa
value: int = driver.source.bb.t2Dvb.get_ldata()
```

Sets the number of data symbols per T2 frame.

return
data_symbols: integer Range: 0 to 4095

get_lf() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:LF
value: int = driver.source.bb.t2Dvb.get_lf()
```

Queries the computed number of OFDM symbols per T2 frame.

return
ofdm_symbols: integer Range: 0 to 4095

get_naux() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NAUX
value: int = driver.source.bb.t2Dvb.get_naux()
```

Queries the number of auxiliary streams. The current firmware does not support auxiliary streams.

return
num_aux_str: integer 0 Fixed response of the query.

get_network_mode() → EnetworkMode

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NETWorkmode
value: enums.EnetworkMode = driver.source.bb.t2Dvb.get_network_mode()
```

Sets the network mode.

return
network_mode: MFN| SFN

get_nsub() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NSUB
value: int = driver.source.bb.t2Dvb.get_nsub()
```

Sets the number of subslices per T2 frame. The number of subslices is '1' for 'T2-MI Interface > Off'.

return
subslices: integer Range: 0 to 32767

get_nt_2_frames() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NT2Frames
value: int = driver.source.bb.t2Dvb.get_nt_2_frames()
```

Sets the number of T2 frames per super frame.

return
nt_2_frames: integer Range: 2 to 255

get_papr() → T2SystemPapr

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PAPR
value: enums.T2SystemPapr = driver.source.bb.t2Dvb.get_papr()
```

Sets the technique to reduce the peak to average power ratio.

return
papr: OFF| TR

get_payload() → PayloadTestStuff

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PAYLoad
value: enums.PayloadTestStuff = driver.source.bb.t2Dvb.get_payload()
```

Defines the payload area content of the packet.

return
payload: PRBS| H00| HFF

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PID
value: int = driver.source.bb.t2Dvb.get_pid()
```

Sets the .

return
pid: integer Range: 0 to 8191

get_pid_test_pack() → PidTestPacket

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PIDTestpack
value: enums.PidTestPacket = driver.source.bb.t2Dvb.get_pid_test_pack()
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

return
pid_ts_packet: NULL| VARiable

get_pilot() → Dvbt2FramingPilotPattern

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PILot
value: enums.Dvbt2FramingPilotPattern = driver.source.bb.t2Dvb.get_pilot()
```

Sets the pilot pattern.

return
pilot_pattern: PP1| PP2| PP3| PP4| PP5| PP6| PP7| PP8

get_profile() → Dvbt2T2SystemProfileMode

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PROFile
value: enums.Dvbt2T2SystemProfileMode = driver.source.bb.t2Dvb.get_profile()
```

Sets the profile mode. Mutes P1FEF, if the modulator operates in a multi profile environment and is used to generate a RF combined T2Base/T2Lite composite signal.

return
profile_mode: SINGLE| MULTI

get_source() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.t2Dvb.get_source()
```

Sets the modulation source for the input signal.

return
dvbt_2_source: EXTernal| TSPLayer| TESTsignal

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:STATe
value: bool = driver.source.bb.t2Dvb.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

get_tfs() → SystemPostExtension

```
# SCPI: [SOURce<HW>]:BB:T2DVb:TFS
value: enums.SystemPostExtension = driver.source.bb.t2Dvb.get_tfs()
```

Queries the state. The current firmware does not support TFS.

return
tfs: OFF OFF Fixed response of the query.

get_ts_packet() → SettingsTestTsPacket

```
# SCPI: [SOURce<HW>]:BB:T2DVb:TSPacket
value: enums.SettingsTestTsPacket = driver.source.bb.t2Dvb.get_ts_packet()
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

return
ts_packet: H184| S187

get_txsys() → Dvbt2Transmission

```
# SCPI: [SOURce<HW>]:BB:T2DVb:TXSYs
value: enums.Dvbt2Transmission = driver.source.bb.t2Dvb.get_txsys()
```

Sets the transmission system.

return
transmission_sys: T2LM| T2LS| NONT2| MISO| SISO T2LM T2 Lite T2LS T2 Lite NONT2 Non-T2 MISO MISO SISO SISO

preset() → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PRESet
driver.source.bb.t2Dvb.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>:BB:T2DVb:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PRESet
driver.source.bb.t2Dvb.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>:BB:T2DVb:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_ldata(*data_symbols: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:LDATa
driver.source.bb.t2Dvb.set_ldata(data_symbols = 1)
```

Sets the number of data symbols per T2 frame.

param data_symbols
integer Range: 0 to 4095

set_naux(*num_aux_str: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NAUX
driver.source.bb.t2Dvb.set_naux(num_aux_str = 1)
```

Queries the number of auxiliary streams. The current firmware does not support auxiliary streams.

param num_aux_str
integer 0 Fixed response of the query.

set_network_mode(*network_mode: EnetworkMode*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NETWorkmode
driver.source.bb.t2Dvb.set_network_mode(network_mode = enums.EnetworkMode.MFN)
```

Sets the network mode.

param network_mode
MFN| SFN

set_nsub(*subslices: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NSUB
driver.source.bb.t2Dvb.set_nsub(subslices = 1)
```

Sets the number of subslices per T2 frame. The number of subslices is '1' for 'T2-MI Interface > Off'.

param subslices
integer Range: 0 to 32767

set_nt_2_frames(*nt_2_frames: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:NT2Frames
driver.source.bb.t2Dvb.set_nt_2_frames(nt_2_frames = 1)
```

Sets the number of T2 frames per super frame.

param nt_2_frames
integer Range: 2 to 255

set_papr(*papr: T2SystemPapr*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PAPR
driver.source.bb.t2Dvb.set_papr(papr = enums.T2SystemPapr.OFF)
```

Sets the technique to reduce the peak to average power ratio.

param papr
OFF| TR

set_payload(*payload: PayloadTestStuff*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PAYLoad
driver.source.bb.t2Dvb.set_payload(payload = enums.PayloadTestStuff.H00)
```

Defines the payload area content of the packet.

param payload
PRBS| H00| HFF

set_pid(*pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PID
driver.source.bb.t2Dvb.set_pid(pid = 1)
```

Sets the .

param pid
integer Range: 0 to 8191

set_pid_test_pack(*pid_ts_packet: PidTestPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PIDTestpack
driver.source.bb.t2Dvb.set_pid_test_pack(pid_ts_packet = enums.PidTestPacket.
↪NULL)
```

If a header is present in the test packet (‘Test TS Packet > Head/184 Payload’) , you can specify a fixed or variable packet identifier (PID) .

param pid_ts_packet
NULL| VARiable

set_pilot(*pilot_pattern: Dvbt2FramingPilotPattern*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PILot
driver.source.bb.t2Dvb.set_pilot(pilot_pattern = enums.Dvbt2FramingPilotPattern.
↪PP1)
```

Sets the pilot pattern.

param pilot_pattern
PP1| PP2| PP3| PP4| PP5| PP6| PP7| PP8

set_profile(*profile_mode: Dvbt2T2SystemProfileMode*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PROFile
driver.source.bb.t2Dvb.set_profile(profile_mode = enums.
↪Dvbt2T2SystemProfileMode.MULTI)
```

Sets the profile mode. Mutes P1FEF, if the modulator operates in a multi profile environment and is used to generate a RF combined T2Base/T2Lite composite signal.

param profile_mode
SINGLE| MULTI

set_source(*dvbt_2_source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SOURce
driver.source.bb.t2Dvb.set_source(dvbt_2_source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param dvbt_2_source
EXTernal| TSPLayer| TESTsignal

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:STATe
driver.source.bb.t2Dvb.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

set_tfs(*tfs: SystemPostExtension*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:TFS
driver.source.bb.t2Dvb.set_tfs(tfs = enums.SystemPostExtension.OFF)
```

Queries the state. The current firmware does not support TFS.

param tfs
OFF OFF Fixed response of the query.

set_ts_packet(*ts_packet: SettingsTestTsPacket*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:TSPacket
driver.source.bb.t2Dvb.set_ts_packet(ts_packet = enums.SettingsTestTsPacket.
↪H184)
```

Specifies the structure of the test transport stream packet that is fed to the modulator.

param ts_packet
H184| S187

set_txsys(*transmission_sys: Dvbt2Transmission*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:TXSYs
driver.source.bb.t2Dvb.set_txsys(transmission_sys = enums.Dvbt2Transmission.
↪MISO)
```

Sets the transmission system.

param transmission_sys
T2LM| T2LS| NONT2| MISO| SISO T2LM T2 Lite T2LS T2 Lite NONT2 Non-T2 MISO MISO SISO SISO

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.clone()
```

Subgroups

6.18.3.24.1 Bandwidth

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:BANDwidth:VARiation
```

class BandwidthCls

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

get_variation() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:BANDwidth:VARiation
value: int = driver.source.bb.t2Dvb.bandwidth.get_variation()
```

Changes the used bandwidth in the range of +/-1000 ppm.

return
bandwidth_var: integer Range: -1000 to 1000, Unit: ppm

set_variation(*bandwidth_var: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:BANDwidth:VARiation
driver.source.bb.t2Dvb.bandwidth.set_variation(bandwidth_var = 1)
```

Changes the used bandwidth in the range of +/-1000 ppm.

param bandwidth_var
integer Range: -1000 to 1000, Unit: ppm

6.18.3.24.2 Channel

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:CHANnel:[BANDwidth]
```

class ChannelCls

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

get_bandwidth() → Dvbt2FramingChannelBandwidth

```
# SCPI: [SOURce<HW>]:BB:T2DVb:CHANnel:[BANDwidth]
value: enums.Dvbt2FramingChannelBandwidth = driver.source.bb.t2Dvb.channel.get_
↪bandwidth()
```

Selects the channel bandwidth.

return
channel_bw: BW_2| BW_5| BW_6| BW_7| BW_8

set_bandwidth(*channel_bw: Dvbt2FramingChannelBandwidth*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:CHANnel:[BANDwidth]
driver.source.bb.t2Dvb.channel.set_bandwidth(channel_bw = enums.
↪Dvbt2FramingChannelBandwidth.BW_2)
```

Selects the channel bandwidth.

param channel_bw
BW_2| BW_5| BW_6| BW_7| BW_8

6.18.3.24.3 Delay

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:DELay:DEViation
[SOURce<HW>]:BB:T2DVb:DELay:DYNamic
[SOURce<HW>]:BB:T2DVb:DELay:MUTep1
[SOURce<HW>]:BB:T2DVb:DELay:PROCess
[SOURce<HW>]:BB:T2DVb:DELay:STATic
[SOURce<HW>]:BB:T2DVb:DELay:TOTal
```

class DelayCls

Delay commands group definition. 12 total commands, 1 Subgroups, 6 group commands

get_deviation() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:DEViation
value: float = driver.source.bb.t2Dvb.delay.get_deviation()
```

Sets the maximum permissible delay.

return
max_dev_time: float Range: 1E-6 to 500E-6

get_dynamic() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:DYNamic
value: float = driver.source.bb.t2Dvb.delay.get_dynamic()
```

Queries the transmission delay currently generated by the SFN delay .

return
dyn_delay: float Range: 0 to 8.0

get_mute_p_1() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:MUTep1
value: bool = driver.source.bb.t2Dvb.delay.get_mute_p_1()
```

Activates muting the P1 symbol of the first T2 frame in a super frame. To mute the P1 symbol, the symbol is set to zero.

return
mute_p_1: 1| ON| 0| OFF

get_process() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:PROCess
value: float = driver.source.bb.t2Dvb.delay.get_process()
```

Queries the delay from the modulator input up to the SFN delay .

return
proc_delay: float Range: 0 to 4.0

get_static() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:STATic
value: float = driver.source.bb.t2Dvb.delay.get_static()
```

Sets the delay to shift the time of transmission positively or negatively.

return
static_delay: float Range: -4.0 to 4.0

get_total() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TOTal
value: float = driver.source.bb.t2Dvb.delay.get_total()
```

Queries the sum of processing delay and dynamic delay.

return
total_delay: float Range: -4.0 to 16.0

set_deviation(*max_dev_time: float*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:DEViation
driver.source.bb.t2Dvb.delay.set_deviation(max_dev_time = 1.0)
```

Sets the maximum permissible delay.

param max_dev_time
float Range: 1E-6 to 500E-6

set_mute_p_1(*mute_p_1: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:MUTep1
driver.source.bb.t2Dvb.delay.set_mute_p_1(mute_p_1 = False)
```

Activates muting the P1 symbol of the first T2 frame in a super frame. To mute the P1 symbol, the symbol is set to zero.

param mute_p_1
1| ON| 0| OFF

set_static(*static_delay: float*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:STATic
driver.source.bb.t2Dvb.delay.set_static(static_delay = 1.0)
```

Sets the delay to shift the time of transmission positively or negatively.

param static_delay
float Range: -4.0 to 4.0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.delay.clone()
```

Subgroups

6.18.3.24.3.1 Tsp

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:DELay:TSP:DATE
[SOURce<HW>]:BB:T2DVb:DELay:TSP:MODE
[SOURce<HW>]:BB:T2DVb:DELay:TSP:OFFSet
[SOURce<HW>]:BB:T2DVb:DELay:TSP:SEConds
[SOURce<HW>]:BB:T2DVb:DELay:TSP:TIME
```

class TspCls

Tsp commands group definition. 6 total commands, 1 Subgroups, 5 group commands

get_date() → str

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:DATE
value: str = driver.source.bb.t2Dvb.delay.tsp.get_date()
```

Queries the UTC date from the last UTC reference update.

return
tsp_date: string Format yyyy-mm-dd

get_mode() → SfnMode

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:MODE
value: enums.SfnMode = driver.source.bb.t2Dvb.delay.tsp.get_mode()
```

Queries the type of the currently received T2-MI timestamps.

return
timestamp_mode: RELative| ABSolute RELative Received T2-MI stream has T2-MI packets with relative timestamps. ABSolute Received T2-MI stream has T2-MI packets with absolute timestamps. If received, the following subparameters are displayed.

get_offset() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:OFFSet
value: int = driver.source.bb.t2Dvb.delay.tsp.get_offset()
```

Modifies the UTC/ leap seconds offset.

return
tsp_offset: integer Range: -255 to 255

get_seconds() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:SEConds
value: int = driver.source.bb.t2Dvb.delay.tsp.get_seconds()
```

Queries the elapsed time in seconds since 2000. The value is based on the value of the last UTC reference update.

return
tsp_seconds: integer Range: 0 to 1099511627775

get_time() → str

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:TIME
value: str = driver.source.bb.t2Dvb.delay.tsp.get_time()
```

Queries the UTC time from the last UTC reference update.

return
tsp_time: string Format hour:minute:second

set_offset(*tsp_offset: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:OFFSet
driver.source.bb.t2Dvb.delay.tsp.set_offset(tsp_offset = 1)
```

Modifies the UTC/ leap seconds offset.

param tsp_offset
integer Range: -255 to 255

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.delay.tsp.clone()
```

Subgroups

6.18.3.24.3.2 Update

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:DELay:TSP:UPDate
```

class UpdateCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:UPDate
driver.source.bb.t2Dvb.delay.tsp.update.set()
```

Triggers an update of the UTC time and date reference.

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

```
# SCPI: [SOURce<HW>]:BB:T2DVb:DELay:TSP:UPDate
driver.source.bb.t2Dvb.delay.tsp.update.set_with_opc()
```

Triggers an update of the UTC time and date reference.

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

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

6.18.3.24.4 Fef

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:FEF:INTerval
[SOURce<HW>]:BB:T2DVb:FEF:LENGth
[SOURce<HW>]:BB:T2DVb:FEF:PAYLoad
[SOURce<HW>]:BB:T2DVb:FEF:TYPE
[SOURce<HW>]:BB:T2DVb:FEF
```

class FefCls

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

get_interval() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:INTerval
value: int = driver.source.bb.t2Dvb.fef.get_interval()
```

Queries the number of T2 frames between two FEF parts. The T2 frame shall always be the first frame in a T2 super frame which contains both FEF parts and T2 frames.

return
fef_interval: integer Range: 0 to 255

get_length() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:LENGth
value: int = driver.source.bb.t2Dvb.fef.get_length()
```

Queries the length of the associated FEF part as the number of elementary periods T, from the start of the P1 symbol of the FEF part to the start of the P1 symbol of the next T2 frame. The FEF length is '0' for 'T2-MI Interface > Off'.

return
fef_length: integer Range: 0 to 16777215

get_payload() → Dvbt2T2SystemFefPayload

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:PAYLoad
value: enums.Dvbt2T2SystemFefPayload = driver.source.bb.t2Dvb.fef.get_payload()
```

Sets the FEF payload.

return
fef_payload: NULL| NOISe NULL I/Q values of the FEF payload are zeroes. NOISe I/Q values of the FEF payload are modulated in the frequency domain using a PRBS and transformed into the time domain by . The technique allows generating payload with a power level equal to the T2 frame.

get_type_py() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:TYPE
value: int = driver.source.bb.t2Dvb.fef.get_type_py()
```

Queries the type of the associated FEF part.

return
fef_type: integer Range: 0 to 15

get_value() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF
value: bool = driver.source.bb.t2Dvb.fef.get_value()
```

Enables/disables .

return
fef: 1| ON| 0| OFF

set_interval(*fef_interval: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:INTerval
driver.source.bb.t2Dvb.fef.set_interval(fef_interval = 1)
```

Queries the number of T2 frames between two FEF parts. The T2 frame shall always be the first frame in a T2 super frame which contains both FEF parts and T2 frames.

param fef_interval
integer Range: 0 to 255

set_length(*fef_length: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:LENGth
driver.source.bb.t2Dvb.fef.set_length(fef_length = 1)
```

Queries the length of the associated FEF part as the number of elementary periods T, from the start of the P1 symbol of the FEF part to the start of the P1 symbol of the next T2 frame. The FEF length is '0' for 'T2-MI Interface > Off'.

param fef_length
integer Range: 0 to 16777215

set_payload(*fef_payload: Dvbt2T2SystemFefPayload*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:PAYLoad
driver.source.bb.t2Dvb.fef.set_payload(fef_payload = enums.
↪Dvbt2T2SystemFefPayload.NOISe)
```

Sets the FEF payload.

param fef_payload
NULL| NOISe NULL I/Q values of the FEF payload are zeroes. NOISe I/Q values of the FEF payload are modulated in the frequency domain using a PRBS and transformed into the time domain by . The technique allows generating payload with a power level equal to the T2 frame.

set_type_py(*fef_type: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF:TYPE
driver.source.bb.t2Dvb.fef.set_type_py(fef_type = 1)
```

Queries the type of the associated FEF part.

param fef_type
integer Range: 0 to 15

set_value(*fef: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FEF
driver.source.bb.t2Dvb.fef.set_value(fef = False)
```

Enables/disables .

param fef
1| ON| 0| OFF

6.18.3.24.5 Fft

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:FFT:MODE
```

class FftCls

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

get_mode() → Dvbt2FramingFftSize

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FFT:MODE
value: enums.Dvbt2FramingFftSize = driver.source.bb.t2Dvb.fft.get_mode()
```

Defines the size.

return
fft_size: M1K| M2K| M4K| M8K| M8E| M16K| M16E| M32K| M32E M1K|M2K|M4K|M8K|M16K|M32K 1K/2K/4K/8K/16K/32K FFT size using normal carrier mode M8E|M16E|M32E 8K/16K/32K FFT size using extended carrier mode

set_mode(*fft_size: Dvbt2FramingFftSize*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:FFT:MODE
driver.source.bb.t2Dvb.fft.set_mode(fft_size = enums.Dvbt2FramingFftSize.M16E)
```

Defines the size.

param fft_size
M1K| M2K| M4K| M8K| M8E| M16K| M16E| M32K| M32E M1K|M2K|M4K|M8K|M16K|M32K 1K/2K/4K/8K/16K/32K FFT size using normal carrier mode M8E|M16E|M32E 8K/16K/32K FFT size using extended carrier mode

6.18.3.24.6 Guard

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:GUARd:INTerval
```

class GuardCls

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

get_interval() → Dvbt2FramingGuardInterval

```
# SCPI: [SOURce<HW>]:BB:T2DVb:GUARd:INTerval
value: enums.Dvbt2FramingGuardInterval = driver.source.bb.t2Dvb.guard.get_
↪interval()
```

Sets the guard interval length.

return
guard_interval: G1_4| G1_8| G1_16| G1_32| G1128| G19128| G19256

set_interval(*guard_interval: Dvbt2FramingGuardInterval*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:GUARd:INTerval
driver.source.bb.t2Dvb.guard.set_interval(guard_interval = enums.
↪Dvbt2FramingGuardInterval.G1_16)
```

Sets the guard interval length.

param guard_interval
G1_4| G1_8| G1_16| G1_32| G1128| G19128| G19256

6.18.3.24.7 Id

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:ID:CELL
[SOURce<HW>]:BB:T2DVb:ID:NETWork
[SOURce<HW>]:BB:T2DVb:ID:T2SYstem
```

class IdCls

Id commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_cell() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:CELL
value: int = driver.source.bb.t2Dvb.id.get_cell()
```

Sets the cell identification (ID) .

return
cell_id: integer 16-bit value in hexadecimal representation. Range: #H0 to #HFFFF

get_network() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:NETWork
value: int = driver.source.bb.t2Dvb.id.get_network()
```

Sets the network identification.

return
network_id: integer 16-bit value in hexadecimal representation. Range: #H0 to #HFFFF

get_t_2_system() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:T2SYstem
value: int = driver.source.bb.t2Dvb.id.get_t_2_system()
```

Sets the T2 system identification.

return
t_2_system_id: integer 16-bit value in hexadecimal representation. Range: #H0 to #HFFFF

set_cell(*cell_id: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:CELL
driver.source.bb.t2Dvb.id.set_cell(cell_id = 1)
```

Sets the cell identification (ID) .

param cell_id
integer 16-bit value in hexadecimal representation. Range: #H0 to #HFFFF

set_network(*network_id: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:NETWork
driver.source.bb.t2Dvb.id.set_network(network_id = 1)
```

Sets the network identification.

param network_id
integer 16-bit value in hexadecimal representation. Range: #H0 to #HFFFF

set_t_2_system(*t_2_system_id: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:T2SYstem
driver.source.bb.t2Dvb.id.set_t_2_system(t_2_system_id = 1)
```

Sets the T2 system identification.

param t_2_system_id
integer 16-bit value in hexadecimal representation. Range: #H0 to #HFFFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.id.clone()
```

Subgroups

6.18.3.24.7.1 Txid

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:ID:TXID:AVAil
```

class TxidCls

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

get_avail() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:TXID:AVAil
value: int = driver.source.bb.t2Dvb.id.txid.get_avail()
```

Queries if transmitter identification signals are available within the current geographic cell.

return
avail: integer 8-bit value in hexadecimal representation. Range: #H0 to #HFF

set_avail(*avail: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:ID:TXID:AVAil
driver.source.bb.t2Dvb.id.txid.set_avail(avail = 1)
```

Queries if transmitter identification signals are available within the current geographic cell.

param avail
integer 8-bit value in hexadecimal representation. Range: #H0 to #HFF

6.18.3.24.8 Info

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:INFO:DP
[SOURce<HW>]:BB:T2DVb:INFO:DPUSed
[SOURce<HW>]:BB:T2DVb:INFO:POSBits
[SOURce<HW>]:BB:T2DVb:INFO:POSCells
[SOURce<HW>]:BB:T2DVb:INFO:PREBits
[SOURce<HW>]:BB:T2DVb:INFO:PRECells
[SOURce<HW>]:BB:T2DVb:INFO:TF
[SOURce<HW>]:BB:T2DVb:INFO:TFEF
[SOURce<HW>]:BB:T2DVb:INFO:TP1
[SOURce<HW>]:BB:T2DVb:INFO:TP2
[SOURce<HW>]:BB:T2DVb:INFO:TS
[SOURce<HW>]:BB:T2DVb:INFO:TSF
```

class InfoCls

Info commands group definition. 12 total commands, 0 Subgroups, 12 group commands

get_dp() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:DP
value: int = driver.source.bb.t2Dvb.info.get_dp()
```

Queries the maximum possible number of PLP data cells in the T2 frame.

return
dplp: integer Range: 0 to 1E7

get_dp_used() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:DPUSed
value: int = driver.source.bb.t2Dvb.info.get_dp_used()
```

Queries the current number of PLP data cells in the T2 frame.

return
dplp_used: integer Range: 0 to 1E7

get_posbits() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:POSBits
value: int = driver.source.bb.t2Dvb.info.get_posbits()
```

Queries the L1-post signaling length in bits.

return
l_1_post_sig_bits: integer Range: 0 to 262175

get_poscells() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:POSCells
value: int = driver.source.bb.t2Dvb.info.get_poscells()
```

Queries the L1-post signaling length in cells.

return
l_1_post_sig_cells: integer Range: 0 to 262143

get_prebits() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:PREBits
value: int = driver.source.bb.t2Dvb.info.get_prebits()
```

Queries the L1-pre signaling length in bits.

return
l_1_pre_sig_bits: integer Range: 0 to 200

get_precells() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:PRECells
value: int = driver.source.bb.t2Dvb.info.get_precells()
```

Queries the L1-pre signaling length in cells.

return
l_1_pre_sig_cells: integer Range: 0 to 1840

get_tf() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:TF
value: float = driver.source.bb.t2Dvb.info.get_tf()
```

Queries the T2 frame duration.

return
t_2_frame_duration: float Range: 0 to 0.999999

get_tfef() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:TFEF
value: int = driver.source.bb.t2Dvb.info.get_tfef()
```

Queries the future extension frame duration.

return
fef_dur: integer Range: 0 to 9.999999

get_tp_1() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:TP1
value: float = driver.source.bb.t2Dvb.info.get_tp_1()
```

Queries the P1 symbol duration.

return
p_1_symbol_dur: float Range: 0 to 0.001000

get_tp_2() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:TP2
value: float = driver.source.bb.t2Dvb.info.get_tp_2()
```

Queries the P2 and data symbol duration.

return
ofdm_symbol_dur: float Range: 0 to 0.010000

get_ts() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:TS
value: float = driver.source.bb.t2Dvb.info.get_ts()
```

Queries the P2 and data symbol duration.

return
ofdm_symbol_dur: float Range: 0 to 0.010000

get_tsf() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INFO:TSF
value: float = driver.source.bb.t2Dvb.info.get_tsf()
```

Queries the super frame duration.

return
super_frame_duration: float Range: 0 to 999.999999

6.18.3.24.9 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:INPut:FORMat
[SOURce<HW>]:BB:T2DVb:INPut:NPLP
[SOURce<HW>]:BB:T2DVb:INPut:TSCHannel
[SOURce<HW>]:BB:T2DVb:INPut
```

class InputPyCls

InputPy commands group definition. 17 total commands, 1 Subgroups, 4 group commands

get_format_py() → CodingInputFormat

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:FORMat
value: enums.CodingInputFormat = driver.source.bb.t2Dvb.inputPy.get_format_py()
```

Sets the input format of the input signal.

return
dvbt_2_inp_format: ASI| SMPTE

get_nplp() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:NPLP
value: int = driver.source.bb.t2Dvb.inputPy.get_nplp()
```

Queries the number of physical layer pipes (PLP) .

return
nplp: integer Range: 1 to 20

get_ts_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:TSCHannel
value: enums.NumberA = driver.source.bb.t2Dvb.inputPy.get_ts_channel()
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

return
dvbt_2_ts_channel: 1| 2| 3| 4

get_value() → CodingInputSignalInputA

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut
value: enums.CodingInputSignalInputA = driver.source.bb.t2Dvb.inputPy.get_
↪value()
```

Sets the external input interface.

return
dvbt_2_input: TS| IP

set_format_py(*dvbt_2_inp_format: CodingInputFormat*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:FORMat
driver.source.bb.t2Dvb.inputPy.set_format_py(dvbt_2_inp_format = enums.
↪CodingInputFormat.ASI)
```

Sets the input format of the input signal.

param dvbt_2_inp_format
ASI| SMPTE

set_ts_channel(*dvbt_2_ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:TSCHannel
driver.source.bb.t2Dvb.inputPy.set_ts_channel(dvbt_2_ts_channel = enums.NumberA.
↪_1)
```

Selects the IP-based transport stream (TS) channel. You can select 1 out of 4 IP TS channels as input at the 'IP Data' interface. To configure a particular channel, see 'IP channel x settings'.

param dvbt_2_ts_channel
1| 2| 3| 4

set_value(*dvbt_2_input: CodingInputSignalInputA*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut
driver.source.bb.t2Dvb.inputPy.set_value(dvbt_2_input = enums.
↪CodingInputSignalInputA.ASI1)
```

Sets the external input interface.

param dvbt_2_input
TS| IP

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.inputPy.clone()
```

Subgroups

6.18.3.24.9.1 T2Mi

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:ANALyzer
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:INTerface
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MEASuremode
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:PID
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:SID
```

class T2MiCls

T2Mi commands group definition. 13 total commands, 3 Subgroups, 5 group commands

get_analyzer() → str

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:ANALyzer
value: str = driver.source.bb.t2Dvb.inputPy.t2Mi.get_analyzer()
```

Queries the status of the T2-MI analyzer by an error message.

return
analyzer: string No error Implies correct behavior of the analyzer.

get_interface() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:INTerface
value: bool = driver.source.bb.t2Dvb.inputPy.t2Mi.get_interface()
```

Activates the T2-MI modulator interface.

return
interface: 1| ON| 0| OFF

get_measure_mode() → Dvbt2InputSignalMeasurementMode

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MEASuremode
value: enums.Dvbt2InputSignalMeasurementMode = driver.source.bb.t2Dvb.inputPy.
↪t2Mi.get_measure_mode()
```

Specifies the measurement mode to configure the evaluation of T2-MI timing parameters.

return
measure_mode: ABSOLUTE| DELTA

get_pid() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:PID
value: int = driver.source.bb.t2Dvb.inputPy.t2Mi.get_pid()
```

Sets the . The PID belongs to MPEG transport stream packets, that contain T2-MI data.

return
pid: integer Range: #H0 to #H1FFF

get_sid() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:SID
value: int = driver.source.bb.t2Dvb.inputPy.t2Mi.get_sid()
```

Sets the T2-MI transport . Use the SID, when transmitting a composite signal, in accordance with annex I of the specification .

return
sid: integer Range: #H0 to #H7

set_interface(*interface: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:INTerface
driver.source.bb.t2Dvb.inputPy.t2Mi.set_interface(interface = False)
```

Activates the T2-MI modulator interface.

param interface
1| ON| 0| OFF

set_measure_mode(*measure_mode: Dvbt2InputSignalMeasurementMode*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MEASuremode
driver.source.bb.t2Dvb.inputPy.t2Mi.set_measure_mode(measure_mode = enums.
↪Dvbt2InputSignalMeasurementMode.ABSOLUTE)
```

Specifies the measurement mode to configure the evaluation of T2-MI timing parameters.

param measure_mode
ABSOLUTE| DELTA

set_pid(*pid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:PID
driver.source.bb.t2Dvb.inputPy.t2Mi.set_pid(pid = 1)
```

Sets the . The PID belongs to MPEG transport stream packets, that contain T2-MI data.

param pid
integer Range: #H0 to #H1FFF

set_sid(*sid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:SID
driver.source.bb.t2Dvb.inputPy.t2Mi.set_sid(sid = 1)
```

Sets the T2-MI transport . Use the SID, when transmitting a composite signal, in accordance with annex I of the specification .

param sid
integer Range: #H0 to #H7

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.inputPy.t2Mi.clone()
```

Subgroups

6.18.3.24.9.2 Max

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T1
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T2
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T3
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T4
```

class MaxCls

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

get_t_1() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T1
value: float = driver.source.bb.t2Dvb.inputPy.t2Mi.max.get_t_1()
```

Queries the current value of the maximum time parameters Tmax1/Tmax2/Tmax3/Tmax4.

return
max_t_1: float Range: -99.999999 to 99.999999, Unit: s

get_t_2() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T2
value: float = driver.source.bb.t2Dvb.inputPy.t2Mi.max.get_t_2()
```

Queries the current value of the maximum time parameters Tmax1/Tmax2/Tmax3/Tmax4.

return
max_t_2: float Range: -99.999999 to 99.999999, Unit: s

get_t_3() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T3
value: float = driver.source.bb.t2Dvb.inputPy.t2Mi.max.get_t_3()
```

Queries the current value of the maximum time parameters Tmax1/Tmax2/Tmax3/Tmax4.

return
max_t_3: float Range: -99.999999 to 99.999999, Unit: s

get_t_4() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MAX:T4
value: float = driver.source.bb.t2Dvb.inputPy.t2Mi.max.get_t_4()
```

Queries the current value of the maximum time parameters Tmax1/Tmax2/Tmax3/Tmax4.

return
max_t_4: float Range: -99.999999 to 99.999999, Unit: s

6.18.3.24.9.3 Min

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MIN:T1
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MIN:T2
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:MIN:T3
```

class MinCls

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

get_t_1() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MIN:T1
value: float = driver.source.bb.t2Dvb.inputPy.t2Mi.min.get_t_1()
```

Queries the current value of minimum time parameters Tmin1/Tmin2/Tmin3.

return
min_t_1: float Range: -99.999999 to 99.999999, Unit: s

get_t_2() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MIN:T2
value: float = driver.source.bb.t2Dvb.inputPy.t2Mi.min.get_t_2()
```

Queries the current value of minimum time parameters Tmin1/Tmin2/Tmin3.

return
min_t_2: float Range: -99.999999 to 99.999999, Unit: s

get_t_3() → float

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:MIN:T3
value: float = driver.source.bb.t2Dvb.inputPy.t2Mi.min.get_t_3()
```

Queries the current value of minimum time parameters Tmin1/Tmin2/Tmin3.

return
min_t_3: float Range: -99.999999 to 99.999999, Unit: s

6.18.3.24.9.4 ResetLog

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:INPut:T2MI:RESetlog
```

class ResetLogCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:RESetlog
driver.source.bb.t2Dvb.inputPy.t2Mi.resetLog.set()
```

Resets the log file.

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

```
# SCPI: [SOURce<HW>]:BB:T2DVb:INPut:T2MI:RESetlog
driver.source.bb.t2Dvb.inputPy.t2Mi.resetLog.set_with_opc()
```

Resets the log file.

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

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

6.18.3.24.10 Lpy

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:L:CONStel
[SOURce<HW>]:BB:T2DVb:L:EXTension
[SOURce<HW>]:BB:T2DVb:L:REPetition
[SOURce<HW>]:BB:T2DVb:L:SCRambled
[SOURce<HW>]:BB:T2DVb:L:T2Baselite
[SOURce<HW>]:BB:T2DVb:L:T2Version
```

class LpyCls

Lpy commands group definition. 8 total commands, 1 Subgroups, 6 group commands

get_constel() → Dvbt2T2SystemL1PostModulation

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:CONStel
value: enums.Dvbt2T2SystemL1PostModulation = driver.source.bb.t2Dvb.lpy.get_
↪constel()
```

Sets the modulation of the L1 post signal.

return
l_1_post_mod: T2| T4| T16| T64 T2 T4 T16 16 T64 64QAM

get_extension() → SystemPostExtension

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:EXTension
value: enums.SystemPostExtension = driver.source.bb.t2Dvb.lpy.get_extension()
```

Queries the L1 post extension state. The current firmware does not support L1 post extension.

return
l_1_post_ext: OFF OFF Fixed response of the query.

get_repetition() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:REPetition
value: bool = driver.source.bb.t2Dvb.lpy.get_repetition()
```

Enables/disables L1 repetition.

return
l_1_repetition: 1| ON| 0| OFF

get_scrambled() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:SCRambled
value: bool = driver.source.bb.t2Dvb.lpy.get_scrambled()
```

Enables/disables L1 post scrambling according to T2 version 1.3.1 of specification . You can query the used version via [:SOURce<hw>]:BB:T2DVb:L:T2Version.

return
l_1_post_scr: 1| ON| 0| OFF

get_t_2_base_lite() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:T2Baselite
value: bool = driver.source.bb.t2Dvb.lpy.get_t_2_base_lite()
```

Enables/disables T2 base lite signaling according to T2 version 1.3.1 of specification . You can query the used version via [:SOURce<hw>]:BB:T2DVb:L:T2Version.

return
t_2_base_lite: 1| ON| 0| OFF

get_t_2_version() → Dvbt2T2SystemL1T2Version

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:T2Version
value: enums.Dvbt2T2SystemL1T2Version = driver.source.bb.t2Dvb.lpy.get_t_2_
↪version()
```

Sets the version of T2 specification , that is used for transmission.

return
t_2_version: V111| V121| V131

set_constel(*l_1_post_mod: Dvbt2T2SystemL1PostModulation*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:CONStel
driver.source.bb.t2Dvb.lpy.set_constel(l_1_post_mod = enums.
↪Dvbt2T2SystemL1PostModulation.T16)
```

Sets the modulation of the L1 post signal.

param l_1_post_mod
T2| T4| T16| T64 T2 T4 T16 16 T64 64QAM

set_extension(*l_1_post_ext: SystemPostExtension*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:EXTension
driver.source.bb.t2Dvb.lpy.set_extension(l_1_post_ext = enums.
↪SystemPostExtension.OFF)
```

Queries the L1 post extension state. The current firmware does not support L1 post extension.

param l_1_post_ext
OFF OFF Fixed response of the query.

set_repetition(*l_1_repetition: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:REPetition
driver.source.bb.t2Dvb.lpy.set_repetition(l_1_repetition = False)
```

Enables/disables L1 repetition.

param l_1_repetition
1| ON| 0| OFF

set_scrambled(*l_1_post_scr: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:SCRambled
driver.source.bb.t2Dvb.lpy.set_scrambled(l_1_post_scr = False)
```

Enables/disables L1 post scrambling according to T2 version 1.3.1 of specification . You can query the used version via [:SOURce<hw>]:BB:T2DVb:L:T2Version.

param l_1_post_scr
1| ON| 0| OFF

set_t_2_base_lite(*t_2_base_lite: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:T2Baselite
driver.source.bb.t2Dvb.lpy.set_t_2_base_lite(t_2_base_lite = False)
```

Enables/disables T2 base lite signaling according to T2 version 1.3.1 of specification . You can query the used version via [:SOURce<hw>]:BB:T2DVb:L:T2Version.

param t_2_base_lite
1| ON| 0| OFF

set_t_2_version(*t_2_version: Dvbt2T2SystemL1T2Version*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:T2Version
driver.source.bb.t2Dvb.lpy.set_t_2_version(t_2_version = enums.
↪Dvbt2T2SystemL1T2Version.V111)
```

Sets the version of T2 specification , that is used for transmission.

param t_2_version
V111| V121| V131

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.lpy.clone()
```

Subgroups

6.18.3.24.10.1 RfSignalling

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:L:RFSignalling:FREQuency
[SOURce<HW>]:BB:T2DVb:L:RFSignalling
```

class RfSignallingCls

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

get_frequency() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:RFSignalling:FREQuency
value: int = driver.source.bb.t2Dvb.lpy.rfSignalling.get_frequency()
```

Queries the signaled frequency in the L1 signaling.

return
l_1_freq: integer Range: 0 to 4294967295, Unit: Hz

get_value() → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:RFSignalling
value: bool = driver.source.bb.t2Dvb.lpy.rfSignalling.get_value()
```

Queries the RF signaling state in L1.
INTRO_CMD_HELP: The setting depends on the setting of the 'T2-MI Interface':

- 'T2-MI Interface > Off': 0x0000 0000 is sent.
- 'T2-MI Interface > On': The value from the T2-MI stream is sent.

return
rf_signalling: 1| ON| 0| OFF

set_value(*rf_signalling: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:L:RFSignalling
driver.source.bb.t2Dvb.lpy.rfSignalling.set_value(rf_signalling = False)
```

Queries the RF signaling state in L1.
INTRO_CMD_HELP: The setting depends on the setting of the 'T2-MI Interface':

- 'T2-MI Interface > Off': 0x0000 0000 is sent.
- 'T2-MI Interface > On': The value from the T2-MI stream is sent.

param rf_signalling
1| ON| 0| OFF

6.18.3.24.11 Miso

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:MISO:MODE
[SOURce<HW>]:BB:T2DVb:MISO:[GROup]
```

class MisoCls

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

get_group() → Dvbt2T2SystemMisoGroupScpi

```
# SCPI: [SOURce<HW>]:BB:T2DVb:MISO:[GROup]
value: enums.Dvbt2T2SystemMisoGroupScpi = driver.source.bb.t2Dvb.miso.get_
↪group()
```

Sets the group.

return
miso_group: G1| G2

get_mode() → AutoManualMode

```
# SCPI: [SOURce<HW>]:BB:T2DVb:MISO:MODE
value: enums.AutoManualMode = driver.source.bb.t2Dvb.miso.get_mode()
```

Sets the group mode, that allows to set the MISO group of the modulator manually.

return
group_mode: MANual

set_group(*miso_group: Dvbt2T2SystemMisoGroupScpi*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:MISO:[GROup]
driver.source.bb.t2Dvb.miso.set_group(miso_group = enums.
↪Dvbt2T2SystemMisoGroupScpi.G1)
```

Sets the group.

param miso_group
G1| G2

6.18.3.24.12 Plp<PhysicalLayerPipe>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.t2Dvb.plp.repcap_physicalLayerPipe_get()
driver.source.bb.t2Dvb.plp.repcap_physicalLayerPipe_set(repcap.PhysicalLayerPipe.Nr1)
```

class PlpCls

Plp commands group definition. 29 total commands, 21 Subgroups, 0 group commands Repeated Capability: PhysicalLayerPipe, default value after init: PhysicalLayerPipe.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.plp.clone()
```

Subgroups

6.18.3.24.12.1 Blocks

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:BLOCks
```

class BlocksCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:BLOCks
value: int = driver.source.bb.t2Dvb.plp.blocks.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries the number of FEC blocks per interleaving frame.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
fec_blocks: integer Range: 0 to 1023

6.18.3.24.12.2 CmType

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:CMTYpe
```

class CmTypeCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → Dvbt2InputSignalCm

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:CMTYpe
value: enums.Dvbt2InputSignalCm = driver.source.bb.t2Dvb.plp.cmType.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Queries the type for multi-PLP. Multi-PLP requires number of PLPs > 1, see [:SOURce<hw>]:BB:T2DVb:INPut:NPLP?.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
cm_type: CCM| ACM CCM Constant coding and modulation. The setting implies identical settings for all s of the commands: [:SOURcehw]:BB:T2DVb:PLPch:FECFrame [:SOURcehw]:BB:T2DVb:PLPch:RATE [:SOURcehw]:BB:T2DVb:PLPch:CONStel [:SOURcehw]:BB:T2DVb:PLPch:CROTation ACM Variable coding and modulation. Not all PLPs use the same coding and modulation.

6.18.3.24.12.3 Constel

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:CONStel
```

class ConstelCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → Dvbt2BicmConstel

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:CONStel
value: enums.Dvbt2BicmConstel = driver.source.bb.t2Dvb.plp.constel.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Defines the constellation.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
constellation: T4| T16| T64| T256 T4 QPSK T16|T64|T256 16/64/256QAM

set(*constellation: Dvbt2BicmConstel*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:CONStel
driver.source.bb.t2Dvb.plp.constel.set(constellation = enums.Dvbt2BicmConstel.
↪T16, physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Defines the constellation.

param constellation
T4| T16| T64| T256 T4 QPSK T16|T64|T256 16/64/256QAM

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.4 Crotation

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:CROTation
```

class CrotationCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:CROTation
value: bool = driver.source.bb.t2Dvb.plp.crotation.get(physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Sets the constellation rotation state.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
crotation: 1| ON| 0| OFF ON Transmits the constellation rotated, i.e. the Q path is delayed vs. the I path. For each constellation, there is a different (but fixed) angle of rotation. OFF Transmits non-rotated constellation.

set(*crotation: bool*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:CROTation
driver.source.bb.t2Dvb.plp.crotation.set(crotation = False, physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Sets the constellation rotation state.

param crotation
1| ON| 0| OFF ON Transmits the constellation rotated, i.e. the Q path is delayed vs. the I path. For each constellation, there is a different (but fixed) angle of rotation. OFF Transmits non-rotated constellation.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.5 FecFrame

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:FECFrame
```

class FecFrameCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → BicmFecFrame

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:FECFrame
value: enums.BicmFecFrame = driver.source.bb.t2Dvb.plp.fecFrame.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Sets the FEC frame.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
np_fec_frame: NORMal| SHORt NORMal NLDPC = 64800 SHORt NLDPC = 16200

set(*np_fec_frame: BicmFecFrame*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:FECFrame
driver.source.bb.t2Dvb.plp.fecFrame.set(np_fec_frame = enums.BicmFecFrame.
↪NORMal, physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Sets the FEC frame.

param np_fec_frame
NORMal| SHORt NORMal NLDPC = 64800 SHORt NLDPC = 16200

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.6 FrameIndex

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:FRAMeindex
```

class FrameIndexCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:FRAMeindex
value: int = driver.source.bb.t2Dvb.plp.frameIndex.get(physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Queries the index of the first frame of the super frame, in that the current PLP occurs.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
ff_index: integer Range: 0 to 255

set(*ff_index: int*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:FRAMeindex
driver.source.bb.t2Dvb.plp.frameIndex.set(ff_index = 1, physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Queries the index of the first frame of the super frame, in that the current PLP occurs.

param ff_index
integer Range: 0 to 255

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.7 Group

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:GROup
```

class GroupCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:GROup
value: int = driver.source.bb.t2Dvb.plp.group.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Sets the PLP group ID for multi-PLP, i.e. the number of PLPs is greater than 1. See [:SOURce<hw>]:BB:T2DVb:INPut:NPLP?.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
group: integer Range: 0 to 255

set(*group: int*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:GROup
driver.source.bb.t2Dvb.plp.group.set(group = 1, physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Sets the PLP group ID for multi-PLP, i.e. the number of PLPs is greater than 1. See [:SOURce<hw>]:BB:T2DVb:INPut:NPLP?.

param group
integer Range: 0 to 255

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.8 Ibs

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:IBS
```

class IbsCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:IBS
value: bool = driver.source.bb.t2Dvb.plp.ibs.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries the in-band signaling state.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
ibs: 1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.plp.ibs.clone()
```

Subgroups

6.18.3.24.12.9 A

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:IBS:A
```

class ACls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:IBS:A
value: bool = driver.source.bb.t2Dvb.plp.ibs.a.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries the in-band signaling type A state. Query requires L1 T2 specification version higher than V1.1.1, see [:SOURce<hw>]:BB:T2DVb:L:T2Version.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
ibsa: 1| ON| 0| OFF

6.18.3.24.12.10 B

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:IBS:B
```

class BCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:IBS:B
value: bool = driver.source.bb.t2Dvb.plp.ibs.b.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries the in-band signaling type B state. Query requires L1 T2 specification version higher than V1.1.1, see [:SOURce<hw>]:BB:T2DVb:L:T2Version.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
ibsb: 1| ON| 0| OFF

6.18.3.24.12.11 Id

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:ID
```

class IdCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:ID
value: int = driver.source.bb.t2Dvb.plp.id.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Sets the PLP ID. The PLP ID has to be unique.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
pl_pid: integer Range: 0 to 255

set(*pl_pid: int*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:ID
driver.source.bb.t2Dvb.plp.id.set(pl_pid = 1, physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Sets the PLP ID. The PLP ID has to be unique.

param pl_pid
integer Range: 0 to 255

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.12 InputPy

class InputPyCls

InputPy 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.bb.t2Dvb.plp.inputPy.clone()
```

Subgroups

6.18.3.24.12.13 DataRate

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:[INPut]:DATarate
```

class DataRateCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:[INPut]:DATarate
value: int = driver.source.bb.t2Dvb.plp.inputPy.dataRate.get(physicalLayerPipe␣
↪= repcap.PhysicalLayerPipe.Default)


    INTRO_CMD_HELP: Queries the measured value of the data rate of one of the␣
↪following:

    - External transport stream including null packets input at 'User 1'␣
↪connector
    - External transport stream including null packets input at 'IP Data/LAN'␣
↪connector (TSoverIP)
```

The value equals the sum of useful data rate rmeas and the rate of null packets r0: rmeas = rmeas + r0

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
data_rate: integer Range: 0 to 999999999

6.18.3.24.12.14 FormatPy

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:FORMat
```

class FormatPyCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → Dvbt2PlpInputFormat

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:FORMat
value: enums.Dvbt2PlpInputFormat = driver.source.bb.t2Dvb.plp.inputPy.formatPy.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Queries the input format of each PLP <num> for all input sources.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
format_py: GFPS| GCS| GSE| TS GFPS Generic fixed-length packetized stream GCS Generic continuous stream GSE Generic stream encapsulation TS Transport stream

set(*format_py: Dvbt2PlpInputFormat*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:FORMat
driver.source.bb.t2Dvb.plp.inputPy.formatPy.set(format_py = enums.
↪Dvbt2PlpInputFormat.GCS, physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Queries the input format of each PLP <num> for all input sources.

param format_py
GFPS| GCS| GSE| TS GFPS Generic fixed-length packetized stream GCS Generic continuous stream GSE Generic stream encapsulation TS Transport stream

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.15 Stuffing

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:STUFfing
```

class StuffingCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:STUFfing
value: bool = driver.source.bb.t2Dvb.plp.inputPy.stuffing.get(physicalLayerPipe␣
↪= repcap.PhysicalLayerPipe.Default)
```

Activates stuffing.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
stuffing: OFF| ON ON Inserts null packets and corrects the values. OFF The data rate of the transport stream source must match the data rate required for the current modulation parameters.

set(*stuffing: bool*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:STUFfing
driver.source.bb.t2Dvb.plp.inputPy.stuffing.set(stuffing = False,␣
↪physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Activates stuffing.

param stuffing
OFF| ON ON Inserts null packets and corrects the values. OFF The data rate of the transport stream source must match the data rate required for the current modulation parameters.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.16 TestSignal

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:TESTsignal
```

class TestSignalCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → CodingInputSignalTestSignal

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:TESTsignal
value: enums.CodingInputSignalTestSignal = driver.source.bb.t2Dvb.plp.inputPy.
↪testSignal.get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Defines the test signal data.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
test_signal: TTSP

set(*test_signal: CodingInputSignalTestSignal*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:INPut:TESTsignal
driver.source.bb.t2Dvb.plp.inputPy.testSignal.set(test_signal = enums.
↪CodingInputSignalTestSignal.TTSP, physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Defines the test signal data.

param test_signal
TTSP

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.17 Issy

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:ISSY
```

class IssyCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → Dvbt2InputIssy

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:ISSY
value: enums.Dvbt2InputIssy = driver.source.bb.t2Dvb.plp.issy.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Queries the state.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
issy: OFF| SHORt| LONG OFF ISSY is not active. ISSY indicator field is 0. SHORt ISSY is active. ISSY indicator field is 1. The synchronizer uses a short . LONG ISSY is active. ISSY indicator field is 1. The synchronizer uses a long .

6.18.3.24.12.18 MaxBlocks

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:MAXBlocks
```

class MaxBlocksCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:MAXBlocks
value: int = driver.source.bb.t2Dvb.plp.maxBlocks.get(physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Queries the maximum number of FEC blocks per interleaving frame.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
fec_blocks_max: integer Range: 0 to 1023

6.18.3.24.12.19 Npd

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:NPD
```

class NpdCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:NPD
value: bool = driver.source.bb.t2Dvb.plp.npd.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries the null packet deletion state.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
npd: 1| ON| 0| OFF

6.18.3.24.12.20 OibPlp

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:OIBPlp
```

class OibPlpCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:OIBPlp
value: int = driver.source.bb.t2Dvb.plp.oibPlp.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries the number of other PLPs signaled within the in-band signaling of the PLP for multi-PLP. Multi-PLP requires number of PLPs > 1, see [:SOURce<hw>]:BB:T2DVb:INPut:NPLP?.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
oib_plps: integer Range: 0 to 255

6.18.3.24.12.21 PacketLength

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:PACKetlength
```

class PacketLengthCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → InputSignalPacketLength

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:PACKetlength
value: enums.InputSignalPacketLength = driver.source.bb.t2Dvb.plp.packetLength.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Queries the packet length of the external transport stream in bytes.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
packet_length: P188| INValid P188 188 byte packets specified for serial input and parallel input. INValid Packet length does not match the specified length.

6.18.3.24.12.22 PadFlag

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:PADFlag
```

class PadFlagCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:PADFlag
value: bool = driver.source.bb.t2Dvb.plp.padFlag.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries if BBFrame padding other than for in-band signaling is used for the current PLP.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
padding_flag: 1| ON| 0| OFF

6.18.3.24.12.23 Rate

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:RATE
```

class RateCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → Dvbt2BicmCoderate

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:RATE
value: enums.Dvbt2BicmCoderate = driver.source.bb.t2Dvb.plp.rate.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Sets the code rate.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
coderate: R1_2| R3_5| R2_3| R3_4| R4_5| R5_6| R1_3| R2_5

set(*coderate: Dvbt2BicmCoderate*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:RATE
driver.source.bb.t2Dvb.plp.rate.set(coderate = enums.Dvbt2BicmCoderate.R1_2,␣
↪physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Sets the code rate.

param coderate
R1_2| R3_5| R2_3| R3_4| R4_5| R5_6| R1_3| R2_5

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.24 StaFlag

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:STAFlag
```

class StaFlagCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → bool

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:STAFlag
value: bool = driver.source.bb.t2Dvb.plp.staFlag.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Queries if the scheduling for the current PLP varies from T2 frame to T2 frame or remains static.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
static_flag: 1| ON| 0| OFF

6.18.3.24.12.25 Til

class TilCls

Til 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.bb.t2Dvb.plp.til.clone()
```

Subgroups

6.18.3.24.12.26 Fint

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:FINT
```

class FintCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:FINT
value: int = driver.source.bb.t2Dvb.plp.til.fint.get(physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Defines the time interleaver frame interval (IJump) . For limitations, see specification .

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
frame_interval: integer Range: 1 to 255

set(*frame_interval: int*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:FINT
driver.source.bb.t2Dvb.plp.til.fint.set(frame_interval = 1, physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Defines the time interleaver frame interval (IJump) . For limitations, see specification .

param frame_interval
integer Range: 1 to 255

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.27 Length

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:LENGth
```

class LengthCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:LENGth
value: int = driver.source.bb.t2Dvb.plp.til.length.get(physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Defines the time interleaver length within the time interleaving frame. For limitations, see specification .

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
til_length: integer Range: 0 to 255

set(*til_length: int*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:LENGth
driver.source.bb.t2Dvb.plp.til.length.set(til_length = 1, physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Defines the time interleaver length within the time interleaving frame. For limitations, see specification .

param til_length
integer Range: 0 to 255

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.28 TypePy

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:TYPE
```

class TypePyCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:TYPE
value: int = driver.source.bb.t2Dvb.plp.til.typePy.get(physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Defines the time interleaver type.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
til_type: integer 0 Maps each interleaving frame directly to a T2 frame. 1 Maps each interleaving frame to more than one T2 frame. Range: 0 to 1

set(*til_type: int*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TIL:TYPE
driver.source.bb.t2Dvb.plp.til.typePy.set(til_type = 1, physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Defines the time interleaver type.

param til_type
integer 0 Maps each interleaving frame directly to a T2 frame. 1 Maps each interleaving frame to more than one T2 frame. Range: 0 to 1

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.29 TypePy

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:TYPE
```

class TypePyCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → Dvbt2ModeStreamAdapterPlpType

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TYPE
value: enums.Dvbt2ModeStreamAdapterPlpType = driver.source.bb.t2Dvb.plp.typePy.
↪get(physicalLayerPipe = repcap.PhysicalLayerPipe.Default)
```

Sets the PLP type. The type depends on the number of PLPs in the setup.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
type_py: DT1| DT2| COMMon COMMon Common PLP of the PLP Group. Requires a multi-PLP setup, see [:SOURcehw]:BB:T2DVb:INPut:NPLP?. DT1 Data type 1.

Fixed for a single-PLP setup. Configurable for a multi-PLP setup. DT2 Data type 2. Requires a multi-PLP setup.

set(*type_py: Dvbt2ModeStreamAdapterPlpType*, *physicalLayerPipe=PhysicalLayerPipe.Default*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:TYPE
driver.source.bb.t2Dvb.plp.typePy.set(type_py = enums.
↪Dvbt2ModeStreamAdapterPlpType.COMMon, physicalLayerPipe = repcap.
↪PhysicalLayerPipe.Default)
```

Sets the PLP type. The type depends on the number of PLPs in the setup.

param type_py
DT1| DT2| COMMon COMMon Common PLP of the PLP Group. Requires a multi-PLP setup, see [:SOURcehw]:BB:T2DVb:INPut:NPLP?. DT1 Data type 1. Fixed for a single-PLP setup. Configurable for a multi-PLP setup. DT2 Data type 2. Requires a multi-PLP setup.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

6.18.3.24.12.30 Useful

class UsefulCls

Useful 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.source.bb.t2Dvb.plp.useful.clone()
```

Subgroups

6.18.3.24.12.31 Rate

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:USEFul:[RATE]
```

class RateCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:USEFul:[RATE]
value: int = driver.source.bb.t2Dvb.plp.useful.rate.get(physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Queries the data rate of useful data ruseful of the external transport stream. The data rate is measured at the input of the installed input interface.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
rate: integer Range: 0 to 999999999

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.t2Dvb.plp.useful.rate.clone()
```

Subgroups

6.18.3.24.12.32 Max

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PLP<CH>:USEFul:[RATE]:MAX
```

class MaxCls

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

get(*physicalLayerPipe=PhysicalLayerPipe.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PLP<CH>:USEFul:[RATE]:MAX
value: int = driver.source.bb.t2Dvb.plp.useful.rate.max.get(physicalLayerPipe =␣
↪repcap.PhysicalLayerPipe.Default)
```

Queries the maximum data rate, that is derived from the current modulation parameter settings. The value is the optimal value at the TS input interface, that is necessary for the modulator.

param physicalLayerPipe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Plp')

return
max_py: integer Range: 0 to 999999999

6.18.3.24.13 Prbs

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:PRBS:[SEQuence]
```

class PrbsCls

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

get_sequence() → SettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PRBS:[SEQuence]
value: enums.SettingsPrbs = driver.source.bb.t2Dvb.prbs.get_sequence()
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

return
prbs: P23_1| P15_1

set_sequence(*prbs: SettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:PRBS:[SEQuence]
driver.source.bb.t2Dvb.prbs.set_sequence(prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence. You can select a PRBS 15 or a PRBS 23 sequence as specified by .

param prbs
P23_1| P15_1

6.18.3.24.14 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:T2DVb:SETTing:CATalog
[SOURce<HW>]:BB:T2DVb:SETTing:DELete
[SOURce<HW>]:BB:T2DVb:SETTing:LOAD
[SOURce<HW>]:BB:T2DVb:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SETTing:DELete
driver.source.bb.t2Dvb.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.dvbt2. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SETTing:CATalog
value: List[str] = driver.source.bb.t2Dvb.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.dvbt2. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
catalog: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SETTing:LOAD
value: str = driver.source.bb.t2Dvb.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dvbt2. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: No help available

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SETTing:STORe
value: str = driver.source.bb.t2Dvb.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.dvbt2) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: No help available

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SETTing:LOAD
driver.source.bb.t2Dvb.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.dvbt2. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
'filename' Filename or complete file path; file extension can be omitted

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:T2DVb:SETTing:STORe
driver.source.bb.t2Dvb.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.dvbt2) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
'filename' Filename or complete file path

6.18.3.24.15 Used

SCPI Command :

```
[SOURce<HW>]:BB:T2DVb:USED:[BANDwidth]
```

class UsedCls

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

get_bandwidth() → int

```
# SCPI: [SOURce<HW>]:BB:T2DVb:USED:[BANDwidth]
value: int = driver.source.bb.t2Dvb.used.get_bandwidth()
```

Queries the used channel bandwidth. The used bandwidth depends on the channel bandwidth, the FFT size and the carrier mode as described in Table 'Dependencies of the used bandwidth'.

return
used_bw: integer Range: 0.0 to 9999999.9

6.18.3.25 Tdmb

SCPI Commands :

```
[SOURce<HW>]:BB:TDMB:ETIinput
[SOURce<HW>]:BB:TDMB:MID
[SOURce<HW>]:BB:TDMB:NET
[SOURce<HW>]:BB:TDMB:NST
[SOURce<HW>]:BB:TDMB:PRESet
[SOURce<HW>]:BB:TDMB:SOURce
[SOURce<HW>]:BB:TDMB:STATe
```

class TdmbCls

Tdmb commands group definition. 34 total commands, 9 Subgroups, 7 group commands

get_eti_input() → TdmbInputSignalEtiSignal

```
# SCPI: [SOURce<HW>]:BB:TDMB:ETIinput
value: enums.TdmbInputSignalEtiSignal = driver.source.bb.tdmb.get_eti_input()
```

Displays whether a valid ETI signal is present and the signal type.

return
eti_signal: INValid| ENI| E559| E537

get_mid() → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:MID
value: int = driver.source.bb.tdmb.get_mid()
```

Displays the DAB mode identity. A mode identity of 0 corresponds to an invalid ETI signal.

return
mode_identity: integer Range: 0 to 4

get_net() → EnetworkMode

```
# SCPI: [SOURce<HW>]:BB:TDMB:NET
value: enums.EnetworkMode = driver.source.bb.tdmb.get_net()
```

Sets the network mode.

return
network_mode: MFN| SFN

get_nst() → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:NST
value: int = driver.source.bb.tdmb.get_nst()
```

Displays the number of streams (NST) contained in the ETI signal.

return
num_of_streams: integer Range: 0 to 64

get_source() → CodingInputSignalSource

```
# SCPI: [SOURce<HW>]:BB:TDMB:SOURce
value: enums.CodingInputSignalSource = driver.source.bb.tdmb.get_source()
```

Sets the modulation source for the input signal.

return
tdmb_source: EXTernal| TSPLayer| TESTsignal

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:TDMB:STATe
value: bool = driver.source.bb.tdmb.get_state()
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

return
state: 1| ON| 0| OFF

preset() → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:PRESet
driver.source.bb.tdmb.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>:BB:TDMB:STATe.

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

```
# SCPI: [SOURce<HW>]:BB:TDMB:PRESet
driver.source.bb.tdmb.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>:BB:TDMB:STATe.

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_net(*network_mode: EnetworkMode*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:NET
driver.source.bb.tdmb.set_net(network_mode = enums.EnetworkMode.MFN)
```

Sets the network mode.

param network_mode
MFN| SFN

set_source(*tdmb_source: CodingInputSignalSource*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:SOURce
driver.source.bb.tdmb.set_source(tdmb_source = enums.CodingInputSignalSource.
↪EXTernal)
```

Sets the modulation source for the input signal.

param tdmb_source
EXTernal| TSPLayer| TESTsignal

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:STATe
driver.source.bb.tdmb.set_state(state = False)
```

Activates the standard and deactivates all the other digital standards and digital modulation modes in the same path.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.18.3.25.1 DataRate<SubChannel>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.source.bb.tdmb.dataRate.repcap_subChannel_get()
driver.source.bb.tdmb.dataRate.repcap_subChannel_set(repcap.SubChannel.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:TDMB:DATarate<CH>
```

class DataRateCls

DataRate commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: SubChannel, default value after init: SubChannel.Nr1

get(*subChannel=SubChannel.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:DATarate<CH>
value: int = driver.source.bb.tdmb.dataRate.get(subChannel = repcap.SubChannel.
↪Default)
```

No command help available

param subChannel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'DataRate')

return
data_rate: integer Range: 0 to 1824

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.tdmb.dataRate.clone()
```

6.18.3.25.2 Delay

SCPI Commands :

```
[SOURce<HW>]:BB:TDMB:DELay:COMPensation
[SOURce<HW>]:BB:TDMB:DELay:DELay
[SOURce<HW>]:BB:TDMB:DELay:DEViation
[SOURce<HW>]:BB:TDMB:DELay:NETWork
[SOURce<HW>]:BB:TDMB:DELay:PROCess
[SOURce<HW>]:BB:TDMB:DELay:STATic
[SOURce<HW>]:BB:TDMB:DELay:TOTal
```

class DelayCls

Delay commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_compensation() → float

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:COMPensation
value: float = driver.source.bb.tdmb.delay.get_compensation()
```

Displays the time span by which signal processing is artificially delayed in order to achieve a constant 'TX Delay'.

return
delay_comp: float Range: -1.0000000 to 1.0000000, Unit: s

get_delay() → float

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:DELay
value: float = driver.source.bb.tdmb.delay.get_delay()
```

Sets the signal turnaround time through the transmitter.

return
delay_tx: float Range: 0.0000000 to 1.0000000, Unit: s

get_deviation() → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:DEViation
value: int = driver.source.bb.tdmb.delay.get_deviation()
```

Sets the maximum permitted deviation of the transmission time relative to the internally regulated reference frequency.

return
delay_dev: integer Range: 0.000001 to 0.0005000, Unit: s

get_network() → float

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:NETWork
value: float = driver.source.bb.tdmb.delay.get_network()
```

Queries the compensating delay. If the delay is added to the network path delay, the overall delay is constant and of known value.

return
network_delay: float Range: 0.0000000 to 1.0000000

get_process() → float

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:PROCess
value: float = driver.source.bb.tdmb.delay.get_process()
```

Queries the minimum signal turnaround time through the transmitter.

return
delay_proc: float Range: 0.0000000 to 1.0000000

get_static() → float

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:STATic
value: float = driver.source.bb.tdmb.delay.get_static()
```

Sets the delay in order to shift the time of transmission positively or negatively.

return
delay_static: float Range: -1.0000000 to 1.0000000, Unit: s

get_total() → float

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:TOTal
value: float = driver.source.bb.tdmb.delay.get_total()
```

Queries the total cycle time of the signal through the transmitter.

return
delay_total: float Range: -1.0000000 to 3.0000000, Unit: s

set_delay(*delay_tx: float*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:DELay
driver.source.bb.tdmb.delay.set_delay(delay_tx = 1.0)
```

Sets the signal turnaround time through the transmitter.

param delay_tx
float Range: 0.0000000 to 1.0000000, Unit: s

set_deviation(*delay_dev: int*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:DEViation
driver.source.bb.tdmb.delay.set_deviation(delay_dev = 1)
```

Sets the maximum permitted deviation of the transmission time relative to the internally regulated reference frequency.

param delay_dev
integer Range: 0.000001 to 0.0005000, Unit: s

set_static(*delay_static: float*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:DELay:STATic
driver.source.bb.tdmb.delay.set_static(delay_static = 1.0)
```

Sets the delay in order to shift the time of transmission positively or negatively.

param delay_static
float Range: -1.0000000 to 1.0000000, Unit: s

6.18.3.25.3 InputPy

SCPI Commands :

```
[SOURce<HW>]:BB:TDMB:INPut:ETIChannel
[SOURce<HW>]:BB:TDMB:INPut:FORMat
[SOURce<HW>]:BB:TDMB:INPut
```

class InputPyCls

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

get_eti_channel() → NumberA

```
# SCPI: [SOURce<HW>]:BB:TDMB:INPut:ETIChannel
value: enums.NumberA = driver.source.bb.tdmb.inputPy.get_eti_channel()
```

Selects the channel that is received over the IP interface.

return
ts_channel: No help available

get_format_py() → TdmbInputSignalInputFormat

```
# SCPI: [SOURce<HW>]:BB:TDMB:INPut:FORMat
value: enums.TdmbInputSignalInputFormat = driver.source.bb.tdmb.inputPy.get_
↪format_py()
```

Sets the format of the input signal.

return
tdmb_format: ETI

get_value() → Atsc30InputType

```
# SCPI: [SOURce<HW>]:BB:TDMB:INPut
value: enums.Atsc30InputType = driver.source.bb.tdmb.inputPy.get_value()
```

Sets the external input interface.

return
tdmb_input: IP| TS

set_eti_channel(*ts_channel: NumberA*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:INPut:ETIChannel
driver.source.bb.tdmb.inputPy.set_eti_channel(ts_channel = enums.NumberA._1)
```

Selects the channel that is received over the IP interface.

param ts_channel
1| 2

set_value(*tdmb_input: Atsc30InputType*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:INPut
driver.source.bb.tdmb.inputPy.set_value(tdmb_input = enums.Atsc30InputType.IP)
```

Sets the external input interface.

param tdmb_input
IP| TS

6.18.3.25.4 Prbs

SCPI Command :

```
[SOURce<HW>]:BB:TDMB:PRBS:[SEQuence]
```

class PrbsCls

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

get_sequence() → TdmbSettingsPrbs

```
# SCPI: [SOURce<HW>]:BB:TDMB:PRBS:[SEQuence]
value: enums.TdmbSettingsPrbs = driver.source.bb.tdmb.prbs.get_sequence()
```

Sets the test signal sequence, that is transmitted in the subchannel. You can select a PRBS 15, PRBS 20 and PRBS 23 sequence as specified by . Also you can define a sequence of zeroes (0x00) . This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

return
prbs: P15_1| P20_1| ZERO| P23_1

set_sequence(*prbs: TdmbSettingsPrbs*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:PRBS:[SEQuence]
driver.source.bb.tdmb.prbs.set_sequence(prbs = enums.TdmbSettingsPrbs.P15_1)
```

Sets the test signal sequence, that is transmitted in the subchannel. You can select a PRBS 15, PRBS 20 and PRBS 23 sequence as specified by . Also you can define a sequence of zeroes (0x00) . This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

param prbs
P15_1| P20_1| ZERO| P23_1

6.18.3.25.5 Protection

class ProtectionCls

Protection 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.bb.tdmb.protection.clone()
```

Subgroups

6.18.3.25.5.1 Level<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.bb.tdmb.protection.level.repcap_index_get()
driver.source.bb.tdmb.protection.level.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:TDMB:PROTection:LEVel<CH>
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → TdmbInputSignalProtectionLevel

```
# SCPI: [SOURce<HW>]:BB:TDMB:PROTection:LEVel<CH>
value: enums.TdmbInputSignalProtectionLevel = driver.source.bb.tdmb.protection.
↪level.get(index = repcap.Index.Default)
```

Queries the protection level. The level depends on the protection profile.

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Level')

return
prot_level: UNDefined| EP1A| EP2A| EP3A| EP4A| EP1B| EP2B| EP3B| EP4B| UP1|

UP2| UP3| UP4| UP5 UP1|UP2|UP3|UP4|UP5 Protection level 1 to 5 with protection profile EP1A|EP2A|EP3A|EP4A Protection level 1A to 4A with protection profile EP1B|EP2B|EP3B|EP4B Protection level 1B to 4B with protection profile UNDefined No protection profile detected

Cloning the Group

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

6.18.3.25.5.2 Profile<Profile>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.bb.tdmb.protection.profile.repcap_profile_get()
driver.source.bb.tdmb.protection.profile.repcap_profile_set(repcap.Profile.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:TDMB:PROTection:PROFile<CH>
```

class ProfileCls

Profile commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Profile, default value after init: Profile.Nr1

get(*profile=Profile.Default*) → TdmbInputSignalProtectionProfile

```
# SCPI: [SOURce<HW>]:BB:TDMB:PROTection:PROFile<CH>
value: enums.TdmbInputSignalProtectionProfile = driver.source.bb.tdmb.
↪protection.profile.get(profile = repcap.Profile.Default)
```

Queries the protection profile.

param profile
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Profile')

return
prot_profile: UEP| EEP UEP Unequal error protection EEP Equal error protection

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.tdmb.protection.profile.clone()
```

6.18.3.25.6 Scid<SubChannel>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.source.bb.tdmb.scid.repcap_subChannel_get()
driver.source.bb.tdmb.scid.repcap_subChannel_set(repcap.SubChannel.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:BB:TDMB:SCID<CH>
```

class ScidCls

Scid commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: SubChannel, default value after init: SubChannel.Nr1

get(*subChannel=SubChannel.Default*) → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:SCID<CH>
value: int = driver.source.bb.tdmb.scid.get(subChannel = repcap.SubChannel.
↪Default)
```

Queries the subchannel identifiers per subchannel.

param subChannel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Scid')

return
sub_channel_id: integer Range: 0 to 63

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bb.tdmb.scid.clone()
```

6.18.3.25.7 Setting

SCPI Commands :

```
[SOURce<HW>]:BB:TDMB:SETTing:CATalog
[SOURce<HW>]:BB:TDMB:SETTing:DELete
[SOURce<HW>]:BB:TDMB:SETTing:LOAD
[SOURce<HW>]:BB:TDMB:SETTing:STORe
```

class SettingCls

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

delete(*delete: str*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:SETTing:DELete
driver.source.bb.tdmb.setting.delete(delete = 'abc')
```

Deletes the selected file from the default or the specified directory. Deleted are files with extension *.tdmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param delete
'filename' Filename or complete file path; file extension can be omitted

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:BB:TDMB:SETTing:CATalog
value: List[str] = driver.source.bb.tdmb.setting.get_catalog()
```

Queries the files with settings in the default directory. Listed are files with the file extension *.tdmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
catalog: filename1,filename2,... Returns a string of filenames separated by commas.

get_load() → str

```
# SCPI: [SOURce<HW>]:BB:TDMB:SETTing:LOAD
value: str = driver.source.bb.tdmb.setting.get_load()
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.tdmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
recall: No help available

get_store() → str

```
# SCPI: [SOURce<HW>]:BB:TDMB:SETTing:STORe
value: str = driver.source.bb.tdmb.setting.get_store()
```

Saves the current settings into the selected file; the file extension (*.tdmb) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

return
save: No help available

set_load(*recall: str*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:SETTing:LOAD
driver.source.bb.tdmb.setting.set_load(recall = 'abc')
```

Loads the selected file from the default or the specified directory. Loaded are files with extension *.tdmb. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param recall
'filename' Filename or complete file path; file extension can be omitted

set_store(*save: str*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:SETTing:STORe
driver.source.bb.tdmb.setting.set_store(save = 'abc')
```

Saves the current settings into the selected file; the file extension (*.tdmb) is assigned automatically. Refer to 'Accessing Files in the Default or Specified Directory' for general information on file handling in the default and in a specific directory.

param save
'filename' Filename or complete file path

6.18.3.25.8 Special

class SpecialCls

Special 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.bb.tdmb.special.clone()
```

Subgroups

6.18.3.25.8.1 Settings

SCPI Command :

```
[SOURce<HW>]:BB:TDMB:[SPECial]:SETTings:[STATe]
```

class SettingsCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:SETTings:[STATe]
value: bool = driver.source.bb.tdmb.special.settings.get_state()
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

return
special_settings: 1| ON| 0| OFF

set_state(*special_settings: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:SETTings:[STATe]
driver.source.bb.tdmb.special.settings.set_state(special_settings = False)
```

Enables/disables special settings. The setting allows you to switch between standard-compliant and user-defined channel coding.

param special_settings
1| ON| 0| OFF

6.18.3.25.8.2 TestSignal

SCPI Commands :

```
[SOURce<HW>]:BB:TDMB:[SPECial]:TESTsignal:SCID
[SOURce<HW>]:BB:TDMB:[SPECial]:TESTsignal:[STATe]
```

class TestSignalCls

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

get_scid() → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TESTsignal:SCID
value: int = driver.source.bb.tdmb.special.testSignal.get_scid()
```

Sets the ID of a subchannel (stream) that transmits a test signal (PRBS) instead of data. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

return
scid: integer Range: 0 to 64

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TESTsignal:[STATe]
value: bool = driver.source.bb.tdmb.special.testSignal.get_state()
```

Activates transfer of a PRBS test signal to a subchannel instead of ETI input data. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

return
prbs_test_signal: 1| ON| 0| OFF

set_scid(*scid: int*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TESTsignal:SCID
driver.source.bb.tdmb.special.testSignal.set_scid(scid = 1)
```

Sets the ID of a subchannel (stream) that transmits a test signal (PRBS) instead of data. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

param scid
integer Range: 0 to 64

set_state(*prbs_test_signal: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TESTsignal:[STATe]
driver.source.bb.tdmb.special.testSignal.set_state(prbs_test_signal = False)
```

Activates transfer of a PRBS test signal to a subchannel instead of ETI input data. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

param prbs_test_signal
1| ON| 0| OFF

6.18.3.25.8.3 Transmission

class TransmissionCls

Transmission 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.source.bb.tdmb.special.transmission.clone()
```

Subgroups

6.18.3.25.8.4 Mode

SCPI Commands :

```
[SOURce<HW>]:BB:TDMB:[SPECial]:TRANsmission:MODE:SELECT
[SOURce<HW>]:BB:TDMB:[SPECial]:TRANsmission:MODE
```

class ModeCls

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

get_select() → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TRANsmission:MODE:SELECT
value: int = driver.source.bb.tdmb.special.transmission.mode.get_select()
```

Selects the transmission mode. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

return
mode_select: integer Range: 1 to 4

get_value() → TdmbSpecialTransmissionMode

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TRANsmission:MODE
value: enums.TdmbSpecialTransmissionMode = driver.source.bb.tdmb.special.
↪transmission.mode.get_value()
```

Sets the transmission mode. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

return
trans_mode: MID| MANual

set_select(*mode_select: int*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TRANsmission:MODE:SELECT
driver.source.bb.tdmb.special.transmission.mode.set_select(mode_select = 1)
```

Selects the transmission mode. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

param mode_select
integer Range: 1 to 4

set_value(*trans_mode: TdmbSpecialTransmissionMode*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:[SPECial]:TRANsmission:MODE
driver.source.bb.tdmb.special.transmission.mode.set_value(trans_mode = enums.
↪TdmbSpecialTransmissionMode.MANual)
```

Sets the transmission mode. This setting takes effect, if special settings are active: SOURce1:BB:TDMB:SPECial:SETTings:STATe 1

param trans_mode
MID| MANual

6.18.3.25.9 Tii

SCPI Commands :

```
[SOURce<HW>]:BB:TDMB:TII:MAIN
[SOURce<HW>]:BB:TDMB:TII:STATe
[SOURce<HW>]:BB:TDMB:TII:SUB
```

class TiiCls

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

get_main() → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:TII:MAIN
value: int = driver.source.bb.tdmb.tii.get_main()
```

Defines the main ID.

return
tii_main: integer Range: 0 to 69

get_state() → bool

```
# SCPI: [SOURce<HW>]:BB:TDMB:TII:STATe
value: bool = driver.source.bb.tdmb.tii.get_state()
```

Enables/disables the transmission of the signal.

return
tii_state: 1| ON| 0| OFF

get_sub() → int

```
# SCPI: [SOURce<HW>]:BB:TDMB:TII:SUB
value: int = driver.source.bb.tdmb.tii.get_sub()
```

Defines the sub ID.

return
tii_sub: integer Range: 1 to 23

set_main(*tii_main: int*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:TII:MAIN
driver.source.bb.tdmb.tii.set_main(tii_main = 1)
```

Defines the main ID.

param tii_main
integer Range: 0 to 69

set_state(*tii_state: bool*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:TII:STATe
driver.source.bb.tdmb.tii.set_state(tii_state = False)
```

Enables/disables the transmission of the signal.

param tii_state
1| ON| 0| OFF

set_sub(*tii_sub: int*) → None

```
# SCPI: [SOURce<HW>]:BB:TDMB:TII:SUB
driver.source.bb.tdmb.tii.set_sub(tii_sub = 1)
```

Defines the sub ID.

param tii_sub
integer Range: 1 to 23

6.18.3.26 Trigger

SCPI Command :

```
[SOURce<HW>]:BB:TRIGger:RMODe
```

class TriggerCls

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

get_rmode() → TrigRunMode

```
# SCPI: [SOURce<HW>]:BB:TRIGger:RMODe
value: enums.TrigRunMode = driver.source.bb.trigger.get_rmode()
```

No command help available

return
running_mode: No help available

6.18.4 Bbin

SCPI Commands :

```
[SOURce<HW>]:BBIN:CDEVice
[SOURce<HW>]:BBIN:CFACtor
[SOURce<HW>]:BBIN:FOFFset
[SOURce<HW>]:BBIN:GIMBalance
[SOURce<HW>]:BBIN:MODE
[SOURce<HW>]:BBIN:MPERiod
[SOURce<HW>]:BBIN:ODELay
[SOURce<HW>]:BBIN:PGAin
[SOURce<HW>]:BBIN:POFFset
[SOURce<HW>]:BBIN:ROUTe
[SOURce<HW>]:BBIN:SKEW
[SOURce<HW>]:BBIN:STATe
```

class BbinCls

Bbin commands group definition. 35 total commands, 8 Subgroups, 12 group commands

get_cdevice() → str

```
# SCPI: [SOURce<HW>]:BBIN:CDEVice
value: str = driver.source.bbin.get_cdevice()
```

Indicates the ID of an externally connected Rohde & Schwarz Instrument or Rohde & Schwarz device.

return
cdevice: string 'None' - no device is connected.

get_cfactor() → float

```
# SCPI: [SOURce<HW>]:BBIN:CFACtor
value: float = driver.source.bbin.get_cfactor()
```

No command help available

return
cfactor: No help available

get_foffset() → float

```
# SCPI: [SOURce<HW>]:BBIN:FOFFset
value: float = driver.source.bbin.get_foffset()
```

Sets a frequency offset for the internal/external baseband signal. The offset affects the generated baseband signal.

return
foffset: float Range: depends on the installed options , Unit: Hz

get_gimbalance() → float

```
# SCPI: [SOURce<HW>]:BBIN:GIMBalance
value: float = driver.source.bbin.get_gimbalance()
```

No command help available

return
gimbalance: No help available

get_mode() → AnalogDigital

```
# SCPI: [SOURce<HW>]:BBIN:MODE
value: enums.AnalogDigital = driver.source.bbin.get_mode()
```

Defines that a digital external signal is applied.

return
mode: DIGital

get_mperiod() → int

```
# SCPI: [SOURce<HW>]:BBIN:MPERiod
value: int = driver.source.bbin.get_mperiod()
```

Sets the recording duration for measuring the baseband input signal by executed [:SOURce<hw>]:BBIN:ALEVel:EXECute.

return
mperiod: integer Range: 1 to 32, Unit: s

get_odelay() → float

```
# SCPI: [SOURce<HW>]:BBIN:ODELay
value: float = driver.source.bbin.get_odelay()
```

No command help available

return
delay: No help available

get_pgain() → float

```
# SCPI: [SOURce<HW>]:BBIN:PGAin
value: float = driver.source.bbin.get_pgain()
```

No command help available

return
pgain: No help available

get_poffset() → float

```
# SCPI: [SOURce<HW>]:BBIN:POFFset
value: float = driver.source.bbin.get_poffset()
```

Sets the relative phase offset for the external baseband signal.

return
poffset: float Range: -999.99 to 999.99, Unit: DEG

get_route() → PathUniCodBbinA

```
# SCPI: [SOURce<HW>]:BBIN:ROUTe
value: enums.PathUniCodBbinA = driver.source.bbin.get_route()
```

Selects the signal route for the internal/external baseband signal.

return
route: A

get_skew() → float

```
# SCPI: [SOURce<HW>]:BBIN:SKEW
value: float = driver.source.bbin.get_skew()
```

No command help available

return
skew: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:BBIN:STATe
value: bool = driver.source.bbin.get_state()
```

Enables feeding of an external digital signal into the signal path.

return
state: 1| ON| 0| OFF

set_cfactor(*cfactor: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:CFACtor
driver.source.bbin.set_cfactor(cfactor = 1.0)
```

No command help available

param cfactor
No help available

set_foffset(*foffset: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:FOFFset
driver.source.bbin.set_foffset(foffset = 1.0)
```

Sets a frequency offset for the internal/external baseband signal. The offset affects the generated baseband signal.

param foffset
float Range: depends on the installed options , Unit: Hz

set_gimbalance(*gimbalance: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:GIMBalance
driver.source.bbin.set_gimbalance(gimbalance = 1.0)
```

No command help available

param gimbalance
No help available

set_mode(*mode: AnalogDigital*) → None

```
# SCPI: [SOURce<HW>]:BBIN:MODE
driver.source.bbin.set_mode(mode = enums.AnalogDigital.ANALog)
```

Defines that a digital external signal is applied.

param mode
DIGital

set_mperiod(*mperiod: int*) → None

```
# SCPI: [SOURce<HW>]:BBIN:MPERiod
driver.source.bbin.set_mperiod(mperiod = 1)
```

Sets the recording duration for measuring the baseband input signal by executed [:SOURce<hw>]:BBIN:ALEVel:EXECute.

param mperiod
integer Range: 1 to 32, Unit: s

set_odelay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:ODELay
driver.source.bbin.set_odelay(delay = 1.0)
```

No command help available

param delay
No help available

set_pgain(*pgain: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:PGAin
driver.source.bbin.set_pgain(pgain = 1.0)
```

No command help available

param pgain
No help available

set_poffset(*poffset: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:POFFset
driver.source.bbin.set_poffset(poffset = 1.0)
```

Sets the relative phase offset for the external baseband signal.

param poffset
float Range: -999.99 to 999.99, Unit: DEG

set_route(*route: PathUniCodBbinA*) → None

```
# SCPI: [SOURce<HW>]:BBIN:ROUTe
driver.source.bbin.set_route(route = enums.PathUniCodBbinA.A)
```

Selects the signal route for the internal/external baseband signal.

param route
A

set_skew(*skew: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:SKEW
driver.source.bbin.set_skew(skew = 1.0)
```

No command help available

param skew
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BBIN:STATe
driver.source.bbin.set_state(state = False)
```

Enables feeding of an external digital signal into the signal path.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.18.4.1 Alevel

class AlevelCls

Alevel 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.bbin.alevel.clone()
```

Subgroups

6.18.4.1.1 Execute

SCPI Command :

```
[SOURce<HW>]:BBIN:ALEVel:EXECute
```

class ExecuteCls

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

set() → None

```
# SCPI: [SOURce<HW>]:BBIN:ALEVel:EXECute
driver.source.bbin.alevel.execute.set()
```

Starts measuring the input signal. The measurement estimates the crest factor, peak and RMS level.

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

```
# SCPI: [SOURce<HW>]:BBIN:ALEVel:EXECute
driver.source.bbin.alevel.execute.set_with_opc()
```

Starts measuring the input signal. The measurement estimates the crest factor, peak and RMS level.

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

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

6.18.4.2 Channel<ChannelNull>

RepCap Settings

```
# Range: Nr0 .. Nr63
rc = driver.source.bbin.channel.repcap_channelNull_get()
driver.source.bbin.channel.repcap_channelNull_set(repcap.ChannelNull.Nr0)
```

class ChannelCls

Channel commands group definition. 7 total commands, 4 Subgroups, 0 group commands Repeated Capability: ChannelNull, default value after init: ChannelNull.Nr0

Cloning the Group

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

Subgroups

6.18.4.2.1 Bb

SCPI Command :

```
[SOURce<HW>]:BBIN:CHANnel<CH0>:BB
```

class BbCls

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

get(*channelNull=ChannelNull.Default*) → BbDigInpBb

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:BB
value: enums.BbDigInpBb = driver.source.bbin.channel.bb.get(channelNull =␣
↪repcap.ChannelNull.Default)
```

No command help available

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbin_iq_hs_ch_bb: No help available

set(*bbin_iq_hs_ch_bb: BbDigInpBb*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:BB
driver.source.bbin.channel.bb.set(bbin_iq_hs_ch_bb = enums.BbDigInpBb.A,␣
↪channelNull = repcap.ChannelNull.Default)
```

No command help available

param bbin_iq_hs_ch_bb
No help available

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.bbin.channel.bb.clone()
```

Subgroups

6.18.4.2.1.1 State

SCPI Command :

```
[SOURce<HW>]:BBIN:CHANnel<CH0>:BB:STATe
```

class StateCls

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

get(*channelNull=ChannelNull.Default*) → bool

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:BB:STATe
value: bool = driver.source.bbin.channel.bb.state.get(channelNull = repcap.
↪ChannelNull.Default)
```

Activates the channel.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbin_iq_hs_chan_sta: 1| ON| 0| OFF

set(*bbin_iq_hs_chan_sta: bool*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:BB:STATe
driver.source.bbin.channel.bb.state.set(bbin_iq_hs_chan_sta = False,␣
↪channelNull = repcap.ChannelNull.Default)
```

Activates the channel.

param bbin_iq_hs_chan_sta
1| ON| 0| OFF

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.4.2.2 Name

SCPI Command :

```
[SOURce<HW>]:BBIN:CHANnel<CH0>:NAME
```

class NameCls

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

get(*channelNull=ChannelNull.Default*) → str

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:NAME
value: str = driver.source.bbin.channel.name.get(channelNull = repcap.
↪ChannelNull.Default)
```

Queries the channel name.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbin_iq_hs_chan_nam: string

set(*bbin_iq_hs_chan_nam: str*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:NAME
driver.source.bbin.channel.name.set(bbin_iq_hs_chan_nam = 'abc', channelNull =␣
↪repcap.ChannelNull.Default)
```

Queries the channel name.

param bbin_iq_hs_chan_nam
string

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.4.2.3 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.source.bbin.channel.power.clone()
```

Subgroups

6.18.4.2.3.1 Cfactor

SCPI Command :

```
[SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:CFACtor
```

class CfactorCls

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

get(*channelNull=ChannelNull.Default*) → float

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:CFACtor
value: float = driver.source.bbin.channel.power.cfactor.get(channelNull =␣
↪repcap.ChannelNull.Default)
```

Sets the crest factor of the individual channels.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbin_iq_hs_ch_cr_fac: float Range: 0 to 30

set(*bbin_iq_hs_ch_cr_fac: float*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:CFACtor
driver.source.bbin.channel.power.cfactor.set(bbin_iq_hs_ch_cr_fac = 1.0,␣
↪channelNull = repcap.ChannelNull.Default)
```

Sets the crest factor of the individual channels.

param bbin_iq_hs_ch_cr_fac
float Range: 0 to 30

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.4.2.3.2 Peak

SCPI Command :

```
[SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:PEAK
```

class PeakCls

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

get(*channelNull=ChannelNull.Default*) → float

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:PEAK
value: float = driver.source.bbin.channel.power.peak.get(channelNull = repcap.
↪ChannelNull.Default)
```

Sets the peak level per channel.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbin_hs_ch_po_peak: float Range: -60 to 3.02

set(*bbin_hs_ch_po_peak: float*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:PEAK
driver.source.bbin.channel.power.peak.set(bbin_hs_ch_po_peak = 1.0, channelNull␣
↪= repcap.ChannelNull.Default)
```

Sets the peak level per channel.

param bbin_hs_ch_po_peak
float Range: -60 to 3.02

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.4.2.3.3 Rms

SCPI Command :

```
[SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:RMS
```

class RmsCls

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

get(*channelNull=ChannelNull.Default*) → float

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:POWer:RMS
value: float = driver.source.bbin.channel.power.rms.get(channelNull = repcap.
↪ChannelNull.Default)
```

Queries the estimated RMS level.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbin_iq_hs_ch_po_rms: float Range: -100 to 10

6.18.4.2.4 SymbolRate

SCPI Command :

```
[SOURce<HW>]:BBIN:CHANnel<CH0>:SRATe
```

class SymbolRateCls

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

get(*channelNull=ChannelNull.Default*) → float

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:SRATe
value: float = driver.source.bbin.channel.symbolRate.get(channelNull = repcap.
↪ChannelNull.Default)
```

Sets the sample rate per channel.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbin_iq_hs_ch_sa_rat: float Range: 400 to 250E6 ('System Config Mode = Advanced') /1250E6 ('System Config Mode = Standard')

set(*bbin_iq_hs_ch_sa_rat: float*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce<HW>]:BBIN:CHANnel<CH0>:SRATe
driver.source.bbin.channel.symbolRate.set(bbin_iq_hs_ch_sa_rat = 1.0,␣
↪channelNull = repcap.ChannelNull.Default)
```

Sets the sample rate per channel.

param bbin_iq_hs_ch_sa_rat
float Range: 400 to 250E6 ('System Config Mode = Advanced') /1250E6 ('System Config Mode = Standard')

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.4.3 Digital

SCPI Command :

```
[SOURce<HW>]:BBIN:DIGital:INTerface
```

class DigitalCls

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

get_interface() → BbinInterfaceMode

```
# SCPI: [SOURce<HW>]:BBIN:DIGital:INTerface
value: enums.BbinInterfaceMode = driver.source.bbin.digital.get_interface()
```

Selects the input connector at that the signal is fed.

return
bbin_dig_interface: DIGital| HSDin | HSDin HSDin Dig. IQ HS 1

set_interface(*bbin_dig_interface: BbinInterfaceMode*) → None

```
# SCPI: [SOURce<HW>]:BBIN:DIGital:INTerface
driver.source.bbin.digital.set_interface(bbin_dig_interface = enums.
↪BbinInterfaceMode.DIGital)
```

Selects the input connector at that the signal is fed.

param bbin_dig_interface
DIGital| HSDin | HSDin HSDin Dig. IQ HS 1

Cloning the Group

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

Subgroups

6.18.4.3.1 Asetting

SCPI Command :

```
[SOURce<HW>]:BBIN:DIGital:ASETting:STATe
```

class AsettingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BBIN:DIGital:ASETting:STATe
value: bool = driver.source.bbin.digital.asetting.get_state()
```

Activates automatic adjustment of the baseband input signal.

return
state: 1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BBIN:DIGital:ASETting:STATe
driver.source.bbin.digital.asetting.set_state(state = False)
```

Activates automatic adjustment of the baseband input signal.

param state
1| ON| 0| OFF

6.18.4.4 Iqswap

SCPI Command :

```
[SOURce<HW>]:BBIN:IQSWap:[STATe]
```

class IqswapCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BBIN:IQSWap:[STATe]
value: bool = driver.source.bbin.iqswap.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:BBIN:IQSWap:[STATe]
driver.source.bbin.iqswap.set_state(state = False)
```

No command help available

param state
No help available

6.18.4.5 Offset

SCPI Commands :

```
[SOURce<HW>]:BBIN:OFFSet:I
[SOURce<HW>]:BBIN:OFFSet:Q
```

class OffsetCls

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

get_icomponent() → float

```
# SCPI: [SOURce<HW>]:BBIN:OFFSet:I
value: float = driver.source.bbin.offset.get_icomponent()
```

No command help available

return
ipart: No help available

get_qcomponent() → float

```
# SCPI: [SOURce<HW>]:BBIN:OFFSet:Q
value: float = driver.source.bbin.offset.get_qcomponent()
```

No command help available

return
qpart: No help available

set_icomponent(*ipart: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:OFFSet:I
driver.source.bbin.offset.set_icomponent(ipart = 1.0)
```

No command help available

param ipart
No help available

set_qcomponent(*qpart: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:OFFSet:Q
driver.source.bbin.offset.set_qcomponent(qpart = 1.0)
```

No command help available

param qpart
No help available

6.18.4.6 Oload

SCPI Command :

```
[SOURce<HW>]:BBIN:OLOad:STATe
```

class OloadCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BBIN:OLOad:STATe
value: bool = driver.source.bbin.oload.get_state()
```

No command help available

return
state: No help available

Cloning the Group

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

Subgroups

6.18.4.6.1 Hold

SCPI Commands :

```
[SOURce<HW>]:BBIN:OLOad:HOLD:RESet
[SOURce<HW>]:BBIN:OLOad:HOLD:STATe
```

class HoldCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:BBIN:OLOad:HOLD:STATe
value: bool = driver.source.bbin.oload.hold.get_state()
```

No command help available

return
state: No help available

reset() → None

```
# SCPI: [SOURce<HW>]:BBIN:OLOad:HOLD:RESet
driver.source.bbin.oload.hold.reset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:BBIN:OLOad:HOLD:RESet
driver.source.bbin.oload.hold.reset_with_opc()
```

No command help available

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

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

6.18.4.7 Power

SCPI Commands :

```
[SOURce<HW>]:BBIN:POWer:CFACtor
[SOURce<HW>]:BBIN:POWer:PEAK
[SOURce<HW>]:BBIN:POWer:RMS
```

class PowerCls

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

get_cfactor() → float

```
# SCPI: [SOURce<HW>]:BBIN:POWer:CFACtor
value: float = driver.source.bbin.power.get_cfactor()
```

Sets the crest factor of the external baseband signal.

return
cfactor: float Range: 0 to 30, Unit: dB

get_peak() → float

```
# SCPI: [SOURce<HW>]:BBIN:POWer:PEAK
value: float = driver.source.bbin.power.get_peak()
```

Peak level of the external baseband signal relative to full scale of 0.5 V (in terms of dB full scale) .

return
peak: float Range: -60 to 3.02, Unit: dBfs

get_rms() → float

```
# SCPI: [SOURce<HW>]:BBIN:POWer:RMS
value: float = driver.source.bbin.power.get_rms()
```

Queries the RMS level of the external digital baseband signal.

return
rms: float Range: -100 to 10

set_cfactor(*cfactor: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:POWer:CFACtor
driver.source.bbin.power.set_cfactor(cfactor = 1.0)
```

Sets the crest factor of the external baseband signal.

param cfactor
float Range: 0 to 30, Unit: dB

set_peak(*peak: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:POWer:PEAK
driver.source.bbin.power.set_peak(peak = 1.0)
```

Peak level of the external baseband signal relative to full scale of 0.5 V (in terms of dB full scale) .

param peak
float Range: -60 to 3.02, Unit: dBfs

6.18.4.8 SymbolRate

SCPI Commands :

```
[SOURce<HW>]:BBIN:SRATe:MAX
[SOURce<HW>]:BBIN:SRATe:SOURce
[SOURce<HW>]:BBIN:SRATe:SUM
[SOURce<HW>]:BBIN:SRATe:[ACTual]
```

class SymbolRateCls

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

get_actual() → float

```
# SCPI: [SOURce<HW>]:BBIN:SRATe:[ACTual]
value: float = driver.source.bbin.symbolRate.get_actual()
```

Queries the sample rate of the external digital baseband signal.

return
actual: float Range: 400 to 100E6

get_max() → int

```
# SCPI: [SOURce<HW>]:BBIN:SRATe:MAX
value: int = driver.source.bbin.symbolRate.get_max()
```

Queries the maximum sample rate.

return
dig_iq_hs_out_sr_max: integer Range: 400 to 600E6

get_source() → BbinSampRateModeb

```
# SCPI: [SOURce<HW>]:BBIN:SRATe:SOURce
value: enums.BbinSampRateModeb = driver.source.bbin.symbolRate.get_source()
```

Queries the digital interface used to estimate the sample rate.

return
source: HSDin HSDin Queried for [:SOURcehw]:BBIN:DIGital:INTerface HSDin.

get_sum() → int

```
# SCPI: [SOURce<HW>]:BBIN:SRATe:SUM
value: int = driver.source.bbin.symbolRate.get_sum()
```

Queries the sum of the sample rates of all active channels.

return
dig_iq_hs_out_sr_sum: integer Range: 0 to depends on settings

set_source(*source: BbinSampRateModeb*) → None

```
# SCPI: [SOURce<HW>]:BBIN:SRATe:SOURce
driver.source.bbin.symbolRate.set_source(source = enums.BbinSampRateModeb.HSDin)
```

Queries the digital interface used to estimate the sample rate.

param source
HSDin HSDin Queried for [:SOURcehw]:BBIN:DIGital:INTerface HSDin.

6.18.5 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

6.18.5.1 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

6.18.5.1.1 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

6.18.5.1.1.1 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.

6.18.5.1.1.2 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

6.18.5.1.1.3 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

6.18.5.1.1.4 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

6.18.5.1.1.5 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

6.18.5.2 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

6.18.5.2.1 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

6.18.5.2.1.1 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() → DexchSepDec

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:DECimal
value: enums.DexchSepDec = 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: DexchSepDec*) → None

```
# SCPI: [SOURce<HW>]:CORRection:DEXChange:AFILe:SEParator:DECimal
driver.source.correction.dexchange.afile.separator.set_decimal(decimal = enums.
↪DexchSepDec.COMMa)
```

Sets the decimal separator used in the ASCII data between '.' (decimal point) and ',' (comma) with floating-point numerals.

param decimal
DOT| COMMa

6.18.5.2.2 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.18.5.3 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

6.18.6 Dm

class DmCls

Dm 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.dm.clone()
```

Subgroups

6.18.6.1 External

class ExternalCls

External 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.dm.external.clone()
```

Subgroups

6.18.6.1.1 Polarity<Channel>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.dm.external.polarity.repcap_channel_get()
driver.source.dm.external.polarity.repcap_channel_set(repcap.Channel.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:DM:EXTernal:POLarity<CH>
```

class PolarityCls

Polarity commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

get(*channel=Channel.Default*) → NormalInverted

```
# SCPI: [SOURce<HW>]:DM:EXTernal:POLarity<CH>
value: enums.NormalInverted = driver.source.dm.external.polarity.get(channel =␣
↪repcap.Channel.Default)
```

No command help available

param channel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Polarity')

return
polarity: No help available

set(*polarity: NormalInverted*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:DM:EXTernal:POLarity<CH>
driver.source.dm.external.polarity.set(polarity = enums.NormalInverted.INVerted,
↪ channel = repcap.Channel.Default)
```

No command help available

param polarity
No help available

param channel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Polarity')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.dm.external.polarity.clone()
```

6.18.6.2 Polarity<Channel>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.dm.polarity.repcap_channel_get()
driver.source.dm.polarity.repcap_channel_set(repcap.Channel.Nr1)
```

SCPI Command :

```
[SOURce<HW>]:DM:POLarity<CH>
```

class PolarityCls

Polarity commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

get(*channel=Channel.Default*) → NormalInverted

```
# SCPI: [SOURce<HW>]:DM:POLarity<CH>
value: enums.NormalInverted = driver.source.dm.polarity.get(channel = repcap.
↪Channel.Default)
```

No command help available

param channel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Polarity')

return
polarity: No help available

set(*polarity: NormalInverted*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:DM:POLarity<CH>
driver.source.dm.polarity.set(polarity = enums.NormalInverted.INVerted, channel␣
↪= repcap.Channel.Default)
```

No command help available

param polarity
No help available

param channel
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Polarity')

Cloning the Group

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

6.18.7 Fm

SCPI Commands :

```
[SOURce<HW>]:FM:SENSitivity
[SOURce<HW>]:FM:[DEViation]
```

class FmCls

Fm commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_deviation() → float

```
# SCPI: [SOURce<HW>]:FM:[DEViation]
value: float = driver.source.fm.get_deviation()
```

No command help available

return
deviation: No help available

get_sensitivity() → float

```
# SCPI: [SOURce<HW>]:FM:SENSitivity
value: float = driver.source.fm.get_sensitivity()
```

No command help available

return
sensitivity: No help available

set_deviation(*deviation: float*) → None

```
# SCPI: [SOURce<HW>]:FM:[DEViation]
driver.source.fm.set_deviation(deviation = 1.0)
```

No command help available

param deviation
No help available

Cloning the Group

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

Subgroups

6.18.7.1 External

SCPI Commands :

```
[SOURce<HW>]:FM:EXTernal:COUPling
[SOURce<HW>]:FM:EXTernal:DEViation
```

class ExternalCls

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

get_coupling() → AcDc

```
# SCPI: [SOURce<HW>]:FM:EXTernal:COUPling
value: enums.AcDc = driver.source.fm.external.get_coupling()
```

No command help available

return
coupling: No help available

get_deviation() → float

```
# SCPI: [SOURce<HW>]:FM:EXTernal:DEViation
value: float = driver.source.fm.external.get_deviation()
```

No command help available

return
deviation: No help available

set_coupling(*coupling: AcDc*) → None

```
# SCPI: [SOURce<HW>]:FM:EXTernal:COUPling
driver.source.fm.external.set_coupling(coupling = enums.AcDc.AC)
```

No command help available

param coupling
No help available

set_deviation(*deviation: float*) → None

```
# SCPI: [SOURce<HW>]:FM:EXTernal:DEViation
driver.source.fm.external.set_deviation(deviation = 1.0)
```

No command help available

param deviation
No help available

6.18.7.2 Internal

SCPI Command :

```
[SOURce<HW>]:FM:INTernal:SOURce
```

class InternalCls

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

get_source() → AmSourceInt

```
# SCPI: [SOURce<HW>]:FM:INTernal:SOURce
value: enums.AmSourceInt = driver.source.fm.internal.get_source()
```

No command help available

return
source: No help available

set_source(*source: AmSourceInt*) → None

```
# SCPI: [SOURce<HW>]:FM:INTernal:SOURce
driver.source.fm.internal.set_source(source = enums.AmSourceInt.LF1)
```

No command help available

param source
No help available

6.18.8 Frequency

SCPI Commands :

```
[SOURce<HW>]:FREQuency:CENTer
[SOURce<HW>]:FREQuency:FREQuency
[SOURce<HW>]:FREQuency:MANual
[SOURce<HW>]:FREQuency:MODE
[SOURce<HW>]:FREQuency:MULTiplier
[SOURce<HW>]:FREQuency:OFFSet
[SOURce<HW>]:FREQuency:SPAN
[SOURce<HW>]:FREQuency:STARt
[SOURce<HW>]:FREQuency:STOP
```

class FrequencyCls

Frequency commands group definition. 15 total commands, 3 Subgroups, 9 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 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'.

get_multiplier() → float

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier
value: float = driver.source.frequency.get_multiplier()
```

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

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 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'.

set_multiplier(*multiplier: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:MULTiplier
driver.source.frequency.set_multiplier(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

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

6.18.8.1 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)

6.18.8.2 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)

6.18.8.3 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].

6.18.9 InputPy

class InputPyCls

InputPy commands group definition. 9 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

6.18.9.1 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

6.18.9.2 User<UserIx>

RepCap Settings

```
# Range: Nr1 .. Nr48
rc = driver.source.inputPy.user.repcap_userIx_get()
driver.source.inputPy.user.repcap_userIx_set(repcap.UserIx.Nr1)
```

class UserCls

User commands group definition. 8 total commands, 4 Subgroups, 0 group commands Repeated Capability: UserIx, default value after init: UserIx.Nr1

Cloning the Group

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

Subgroups

6.18.9.2.1 Clock

SCPI Commands :

```
[SOURce]:INPut:USER:CLOCk:IMPedance
[SOURce]:INPut:USER:CLOCk:LEVel
[SOURce]:INPut:USER:CLOCk:SLOPe
```

class ClockCls

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

get_impedance() → ImpG50G1KcoerceG10K

```
# SCPI: [SOURce]:INPut:USER:CLOCk:IMPedance
value: enums.ImpG50G1KcoerceG10K = driver.source.inputPy.user.clock.get_
↪impedance()
```

Selects the input impedance for the external trigger inputs.

return
impedance: G50| G1K

get_level() → float

```
# SCPI: [SOURce]:INPut:USER:CLOCk:LEVel
value: float = driver.source.inputPy.user.clock.get_level()
```

Sets the threshold for any input signal at the User1-2 connectors.

return
level: float Range: 0.1 to 2

get_slope() → SlopeType

```
# SCPI: [SOURce]:INPut:USER:CLOCk:SLOPe
value: enums.SlopeType = driver.source.inputPy.user.clock.get_slope()
```

Sets the polarity of the active slope of an externally applied clock signal.

return
slope: NEGative| POSitive

set_impedance(*impedance: ImpG50G1KcoerceG10K*) → None

```
# SCPI: [SOURce]:INPut:USER:CLOCk:IMPedance
driver.source.inputPy.user.clock.set_impedance(impedance = enums.
↪ImpG50G1KcoerceG10K.G1K)
```

Selects the input impedance for the external trigger inputs.

param impedance
G50| G1K

set_level(*level: float*) → None

```
# SCPI: [SOURce]:INPut:USER:CLOCk:LEVel
driver.source.inputPy.user.clock.set_level(level = 1.0)
```

Sets the threshold for any input signal at the User1-2 connectors.

param level
float Range: 0.1 to 2

set_slope(*slope: SlopeType*) → None

```
# SCPI: [SOURce]:INPut:USER:CLOCk:SLOPe
driver.source.inputPy.user.clock.set_slope(slope = enums.SlopeType.NEGative)
```

Sets the polarity of the active slope of an externally applied clock signal.

param slope
NEGative| POSitive

6.18.9.2.2 Direction

SCPI Command :

```
[SOURce]:INPut:USER<CH>:DIRection
```

class DirectionCls

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

get(*userIx=UserIx.Default*) → ConnDirection

```
# SCPI: [SOURce]:INPut:USER<CH>:DIRection
value: enums.ConnDirection = driver.source.inputPy.user.direction.get(userIx =␣
↪repcap.UserIx.Default)
```

Sets the direction of the signal at the connector that can be input or an output.

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

return
direction: INPut| OUTPut| UNUSed INPut|OUTPut Input signal or output signal UNUSed No signal present at the connector.

set(*direction: ConnDirection*, *userIx=UserIx.Default*) → None

```
# SCPI: [SOURce]:INPut:USER<CH>:DIRection
driver.source.inputPy.user.direction.set(direction = enums.ConnDirection.INPut,␣
↪userIx = repcap.UserIx.Default)
```

Sets the direction of the signal at the connector that can be input or an output.

param direction
INPut| OUTPut| UNUSed INPut|OUTPut Input signal or output signal UNUSed No signal present at the connector.

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

6.18.9.2.3 Signal

SCPI Command :

```
[SOURce]:INPut:USER<CH>:SIGNal
```

class SignalCls

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

get(*userIx=UserIx.Default*) → InpOutpConnGlbMapSignb

```
# SCPI: [SOURce]:INPut:USER<CH>:SIGNal
value: enums.InpOutpConnGlbMapSignb = driver.source.inputPy.user.signal.
↪get(userIx = repcap.UserIx.Default)
```

Determines the control signal that is input at the selected connector. To define the connector direction, use the command [:SOURce]:INPut:USER<ch>:DIRection.

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

return
signal: TRIG1| NSEGM1| INST| TS| ETI| SDIF| PPS TRIG1 Global trigger input signal available at 'User 1/2' connector NSEGM1 Input global next segment for triggering of multi-segment waveform files. The signal is available at 'User 1/2' connector. INST Internal instrument trigger signal available at 'User 1/2' connector. TS Transport stream (TS) input signal available at 'User 1' connector only ETI Ensemble transport interface input signal compatible with DAB/T-DMB ETSI standard. The signal is available at 'User 1' connector only. SDIF S/PDIF input signal available at 'User 1' connector only PPS 1PPS (one pulse per second) input signal available at 'User 2' connector only

set(*signal: InpOutpConnGlbMapSignb*, *userIx=UserIx.Default*) → None

```
# SCPI: [SOURce]:INPut:USER<CH>:SIGNal
driver.source.inputPy.user.signal.set(signal = enums.InpOutpConnGlbMapSignb.
↪CLOCK1, userIx = repcap.UserIx.Default)
```

Determines the control signal that is input at the selected connector. To define the connector direction, use the command [:SOURce]:INPut:USER<ch>:DIRection.

param signal
TRIG1| NSEGM1| INST| TS| ETI| SDIF| PPS TRIG1 Global trigger input signal available at 'User 1/2' connector NSEGM1 Input global next segment for triggering of multi-segment waveform files. The signal is available at 'User 1/2' connector. INST Internal instrument trigger signal available at 'User 1/2' connector. TS Transport

stream (TS) input signal available at 'User 1' connector only ETI Ensemble transport interface input signal compatible with DAB/T-DMB ETSI standard. The signal is available at 'User 1' connector only. SDIF S/PDIF input signal available at 'User 1' connector only PPS 1PPS (one pulse per second) input signal available at 'User 2' connector only

param userIx
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

6.18.9.2.4 Trigger

SCPI Commands :

```
[SOURce]:INPut:USER:TRIGger:IMPedance
[SOURce]:INPut:USER:TRIGger:LEVel
[SOURce]:INPut:USER:TRIGger:SLOPe
```

class TriggerCls

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

get_impedance() → ImpG50G1KcoerceG10K

```
# SCPI: [SOURce]:INPut:USER:TRIGger:IMPedance
value: enums.ImpG50G1KcoerceG10K = driver.source.inputPy.user.trigger.get_
↪impedance()
```

Selects the input impedance for the external trigger inputs.

return
impedance: G50| G1K

get_level() → float

```
# SCPI: [SOURce]:INPut:USER:TRIGger:LEVel
value: float = driver.source.inputPy.user.trigger.get_level()
```

Sets the threshold for any input signal at the User1-2 connectors.

return
level: float Range: 0.1 to 2

get_slope() → SlopeType

```
# SCPI: [SOURce]:INPut:USER:TRIGger:SLOPe
value: enums.SlopeType = driver.source.inputPy.user.trigger.get_slope()
```

Sets the polarity of the active slope of an applied instrument trigger.

return
slope: NEGative| POSitive

set_impedance(*impedance: ImpG50G1KcoerceG10K*) → None

```
# SCPI: [SOURce]:INPut:USER:TRIGger:IMPedance
driver.source.inputPy.user.trigger.set_impedance(impedance = enums.
↪ImpG50G1KcoerceG10K.G1K)
```

Selects the input impedance for the external trigger inputs.

param impedance
G50| G1K

set_level(*level: float*) → None

```
# SCPI: [SOURce]:INPut:USER:TRIGger:LEVel
driver.source.inputPy.user.trigger.set_level(level = 1.0)
```

Sets the threshold for any input signal at the User1-2 connectors.

param level
float Range: 0.1 to 2

set_slope(*slope: SlopeType*) → None

```
# SCPI: [SOURce]:INPut:USER:TRIGger:SLOPe
driver.source.inputPy.user.trigger.set_slope(slope = enums.SlopeType.NEGative)
```

Sets the polarity of the active slope of an applied instrument trigger.

param slope
NEGative| POSitive

6.18.10 Iq

SCPI Commands :

```
[SOURce<HW>]:IQ:CREStfactor
[SOURce<HW>]:IQ:SOURce
[SOURce<HW>]:IQ:STATe
[SOURce<HW>]:IQ:WBSTate
```

class IqCls

Iq commands group definition. 140 total commands, 4 Subgroups, 4 group commands

get_crest_factor() → float

```
# SCPI: [SOURce<HW>]:IQ:CREStfactor
value: float = driver.source.iq.get_crest_factor()
```

No command help available

return
crest_factor: No help available

get_source() → IqMode

```
# SCPI: [SOURce<HW>]:IQ:SOURce
value: enums.IqMode = driver.source.iq.get_source()
```

No command help available

return
source: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:STATe
value: bool = driver.source.iq.get_state()
```

Enables/disables the I/Q modulation.

return
state: 1| ON| 0| OFF

get_wb_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:WBSTate
value: bool = driver.source.iq.get_wb_state()
```

No command help available

return
wb_state: No help available

set_crest_factor(*crest_factor: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:CRESTfactor
driver.source.iq.set_crest_factor(crest_factor = 1.0)
```

No command help available

param crest_factor
No help available

set_source(*source: IqMode*) → None

```
# SCPI: [SOURce<HW>]:IQ:SOURce
driver.source.iq.set_source(source = enums.IqMode.ANALog)
```

No command help available

param source
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:STATe
driver.source.iq.set_state(state = False)
```

Enables/disables the I/Q modulation.

param state
1| ON| 0| OFF

set_wb_state(*wb_state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:WBSTate
driver.source.iq.set_wb_state(wb_state = False)
```

No command help available

param wb_state
No help available

Cloning the Group

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

Subgroups

6.18.10.1 Dpd

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:AMFirst
[SOURce<HW>]:IQ:DPD:LREFerence
[SOURce<HW>]:IQ:DPD:PRESet
[SOURce<HW>]:IQ:DPD:STATe
```

class DpdCls

Dpd commands group definition. 47 total commands, 9 Subgroups, 4 group commands

get_am_first() → bool

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMFirst
value: bool = driver.source.iq.dpd.get_am_first()
```

No command help available

return
am_am_first_state: No help available

get_lreference() → DpdPowRef

```
# SCPI: [SOURce<HW>]:IQ:DPD:LREFerence
value: enums.DpdPowRef = driver.source.iq.dpd.get_lreference()
```

No command help available

return
level_reference: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:DPD:STATe
value: bool = driver.source.iq.dpd.get_state()
```

No command help available

return
state: No help available

preset() → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:PRESet
driver.source.iq.dpd.preset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:IQ:DPD:PRESet
driver.source.iq.dpd.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_am_first(*am_am_first_state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMFirst
driver.source.iq.dpd.set_am_first(am_am_first_state = False)
```

No command help available

param am_am_first_state
No help available

set_lreference(*level_reference: DpdPowRef*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:LREFerence
driver.source.iq.dpd.set_lreference(level_reference = enums.DpdPowRef.ADPD)
```

No command help available

param level_reference
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:STATe
driver.source.iq.dpd.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.iq.dpd.clone()
```

Subgroups

6.18.10.1.1 Amam

SCPI Command :

```
[SOURce<HW>]:IQ:DPD:AMAM:STATe
```

class AmamCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMAM:STATe
value: bool = driver.source.iq.dpd.amam.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMAM:STATe
driver.source.iq.dpd.amam.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.iq.dpd.amam.clone()
```

Subgroups

6.18.10.1.1.1 Value

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:AMAM:VALue
[SOURce<HW>]:IQ:DPD:AMAM:VALue:LEVel
[SOURce<HW>]:IQ:DPD:AMAM:VALue:PEP
```

class ValueCls

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

get(*xvalue: float*, *xunit: Unknown*) → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMAM:VALue
value: float = driver.source.iq.dpd.amam.value.get(xvalue = 1.0, xunit = enums.
↪Unknown.DBM)
```

No command help available

param xvalue
No help available

param xunit
No help available

return
delta_power: No help available

get_level() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMAM:VALue:LEVel
value: float = driver.source.iq.dpd.amam.value.get_level()
```

No command help available

return
delta_power: No help available

get_pep() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMAM:VALue:PEP
value: float = driver.source.iq.dpd.amam.value.get_pep()
```

No command help available

return
delta_power: No help available

6.18.10.1.2 AmPm

SCPI Command :

```
[SOURce<HW>]:IQ:DPD:AMPM:STATe
```

class AmPmCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMPM:STATe
value: bool = driver.source.iq.dpd.amPm.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMPM:STATe
driver.source.iq.dpd.amPm.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.iq.dpd.amPm.clone()
```

Subgroups

6.18.10.1.2.1 Value

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:AMPM:VALue
[SOURce<HW>]:IQ:DPD:AMPM:VALue:LEVel
[SOURce<HW>]:IQ:DPD:AMPM:VALue:PEP
```

class ValueCls

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

get(*xvalue: float*, *xunit: Unknown*) → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMPM:VALue
value: float = driver.source.iq.dpd.amPm.value.get(xvalue = 1.0, xunit = enums.
↪Unknown.DBM)
```

No command help available

param xvalue
No help available

param xunit
No help available

return
delta_phase: No help available

get_level() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMPM:VALue:LEVel
value: float = driver.source.iq.dpd.amPm.value.get_level()
```

No command help available

return
delta_phase: No help available

get_pep() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:AMPM:VALue:PEP
value: float = driver.source.iq.dpd.amPm.value.get_pep()
```

No command help available

return
delta_phase: No help available

6.18.10.1.3 Gain

SCPI Command :

```
[SOURce<HW>]:IQ:DPD:GAIN:PRE
```

class GainCls

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

get_pre() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:GAIN:PRE
value: float = driver.source.iq.dpd.gain.get_pre()
```

No command help available

return
pre_gain: No help available

set_pre(*pre_gain: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:GAIN:PRE
driver.source.iq.dpd.gain.set_pre(pre_gain = 1.0)
```

No command help available

param pre_gain
No help available

6.18.10.1.4 InputPy

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:INPut:CFACtor
[SOURce<HW>]:IQ:DPD:INPut:LEVel
[SOURce<HW>]:IQ:DPD:INPut:PEP
```

class InputPyCls

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

get_cfactor() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:INPut:CFACtor
value: float = driver.source.iq.dpd.inputPy.get_cfactor()
```

No command help available

return
crest_factor: No help available

get_level() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:INPut:LEVel
value: float = driver.source.iq.dpd.inputPy.get_level()
```

No command help available

return
level: No help available

get_pep() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:INPut:PEP
value: float = driver.source.iq.dpd.inputPy.get_pep()
```

No command help available

return
pep: No help available

6.18.10.1.5 Measurement

SCPI Command :

```
[SOURce<HW>]:IQ:DPD:MEASurement:STATe
```

class MeasurementCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:DPD:MEASurement:STATe
value: bool = driver.source.iq.dpd.measurement.get_state()
```

No command help available

return
measure_validity: No help available

6.18.10.1.6 Output

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:OUTPut:CFACtor
[SOURce<HW>]:IQ:DPD:OUTPut:LEVel
[SOURce<HW>]:IQ:DPD:OUTPut:PEP
```

class OutputCls

Output commands group definition. 6 total commands, 2 Subgroups, 3 group commands

get_cfactor() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:CFACtor
value: float = driver.source.iq.dpd.output.get_cfactor()
```

No command help available

return
crest_factor: No help available

get_level() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:LEVel
value: float = driver.source.iq.dpd.output.get_level()
```

No command help available

return
level: No help available

get_pep() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:PEP
value: float = driver.source.iq.dpd.output.get_pep()
```

No command help available

return
pep: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.dpd.output.clone()
```

Subgroups

6.18.10.1.6.1 Error

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:OUTPut:ERRor:MAX
[SOURce<HW>]:IQ:DPD:OUTPut:ERRor
```

class ErrorCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:ERRor:MAX
value: float = driver.source.iq.dpd.output.error.get_max()
```

No command help available

return
maximum_error: No help available

get_value() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:ERRor
value: float = driver.source.iq.dpd.output.error.get_value()
```

No command help available

return
achieved_error: No help available

set_max(*maximum_error: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:ERRor:MAX
driver.source.iq.dpd.output.error.set_max(maximum_error = 1.0)
```

No command help available

param maximum_error
No help available

6.18.10.1.6.2 Iterations

SCPI Command :

```
[SOURce<HW>]:IQ:DPD:OUTPut:ITERations:MAX
```

class IterationsCls

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

get_max() → int

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:ITERations:MAX
value: int = driver.source.iq.dpd.output.iterations.get_max()
```

No command help available

return
max_iterations: No help available

set_max(*max_iterations: int*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:OUTPut:ITERations:MAX
driver.source.iq.dpd.output.iterations.set_max(max_iterations = 1)
```

No command help available

param max_iterations
No help available

6.18.10.1.7 Pin

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:PIN:MAX
[SOURce<HW>]:IQ:DPD:PIN:MIN
```

class PinCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:PIN:MAX
value: float = driver.source.iq.dpd.pin.get_max()
```

No command help available

return
pep_in_max: No help available

get_min() → float

```
# SCPI: [SOURce<HW>]:IQ:DPD:PIN:MIN
value: float = driver.source.iq.dpd.pin.get_min()
```

No command help available

return
pep_in_min: No help available

set_max(*pep_in_max: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:PIN:MAX
driver.source.iq.dpd.pin.set_max(pep_in_max = 1.0)
```

No command help available

param pep_in_max
No help available

set_min(*pep_in_min: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:PIN:MIN
driver.source.iq.dpd.pin.set_min(pep_in_min = 1.0)
```

No command help available

param pep_in_min
No help available

6.18.10.1.8 Setting

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:SETTing:CATalog
[SOURce<HW>]:IQ:DPD:SETTing:LOAD
[SOURce<HW>]:IQ:DPD:SETTing:STORe
```

class SettingCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:DPD:SETTing:CATalog
value: List[str] = driver.source.iq.dpd.setting.get_catalog()
```

No command help available

return
catalog: No help available

load(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SETTing:LOAD
driver.source.iq.dpd.setting.load(filename = 'abc')
```

No command help available

param filename
No help available

set_store(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SETTing:STORe
driver.source.iq.dpd.setting.set_store(filename = 'abc')
```

No command help available

param filename
No help available

6.18.10.1.9 Shaping

SCPI Command :

```
[SOURce<HW>]:IQ:DPD:SHAPing:MODE
```

class ShapingCls

Shaping commands group definition. 19 total commands, 3 Subgroups, 1 group commands

get_mode() → DpdShapeMode

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:MODE
value: enums.DpdShapeMode = driver.source.iq.dpd.shaping.get_mode()
```

No command help available

return
shaping: No help available

set_mode(*shaping: DpdShapeMode*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:MODE
driver.source.iq.dpd.shaping.set_mode(shaping = enums.DpdShapeMode.NORMalized)
```

No command help available

param shaping
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.dpd.shaping.clone()
```

Subgroups

6.18.10.1.9.1 Normalized

class NormalizedCls

Normalized 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.iq.dpd.shaping.normalized.clone()
```

Subgroups

6.18.10.1.9.2 Data

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA:CATalog
[SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA:LOAD
[SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA:STORe
[SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA
```

class DataCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA:CATalog
value: List[str] = driver.source.iq.dpd.shaping.normalized.data.get_catalog()
```

No command help available

return
catalog: No help available

get_value() → bytes

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA
value: bytes = driver.source.iq.dpd.shaping.normalized.data.get_value()
```

No command help available

return
data: No help available

load(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA:LOAD
driver.source.iq.dpd.shaping.normalized.data.load(filename = 'abc')
```

No command help available

param filename
No help available

set_store(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA:STORe
driver.source.iq.dpd.shaping.normalized.data.set_store(filename = 'abc')
```

No command help available

param filename
No help available

set_value(*data: bytes*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:NORMalized:DATA
driver.source.iq.dpd.shaping.normalized.data.set_value(data = b'ABCDEFGH')
```

No command help available

param data
No help available

6.18.10.1.9.3 Polynomial

class PolynomialCls

Polynomial 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.iq.dpd.shaping.polynomial.clone()
```

Subgroups

6.18.10.1.9.4 Coefficients

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients
[SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients:CATalog
[SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients:LOAD
[SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients:STORe
```

class CoefficientsCls

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

class CoefficientsStruct

Response structure. Fields:

- Ipart_0: List[float]: No parameter help available
- J_0: float: No parameter help available
- I_1: float: No parameter help available
- J_1: float: No parameter help available

get() → CoefficientsStruct

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients
value: CoefficientsStruct = driver.source.iq.dpd.shaping.polynomial.
↪coefficients.get()
```

No command help available

return
structure: for return value, see the help for CoefficientsStruct structure arguments.

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients:CATalog
value: List[str] = driver.source.iq.dpd.shaping.polynomial.coefficients.get_
↪catalog()
```

No command help available

return
catalog: No help available

load(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients:LOAD
driver.source.iq.dpd.shaping.polynomial.coefficients.load(filename = 'abc')
```

No command help available

param filename
No help available

set(*ipart_0: List[float]*, *j_0: float*, *i_1: float*, *j_1: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients
driver.source.iq.dpd.shaping.polynomial.coefficients.set(ipart_0 = [1.1, 2.2, 3.
↪3], j_0 = 1.0, i_1 = 1.0, j_1 = 1.0)
```

No command help available

param ipart_0
No help available

param j_0
No help available

param i_1
No help available

param j_1
No help available

set_store(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:POLYnomial:COEFficients:STORe
driver.source.iq.dpd.shaping.polynomial.coefficients.set_store(filename = 'abc')
```

No command help available

param filename
No help available

6.18.10.1.9.5 Table

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:INTerp
[SOURce<HW>]:IQ:DPD:SHAPing:[TABLe]:INVert
```

class TableCls

Table commands group definition. 10 total commands, 2 Subgroups, 2 group commands

get_interp() → IqOutEnvInterp

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:INTerp
value: enums.IqOutEnvInterp = driver.source.iq.dpd.shaping.table.get_interp()
```

No command help available

return
ipart_interpolation: No help available

get_invert() → bool

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:[TABLe]:INVert
value: bool = driver.source.iq.dpd.shaping.table.get_invert()
```

No command help available

return
ipart_invert_values: No help available

set_interp(*ipart_interpolation: IqOutEnvInterp*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:INTerp
driver.source.iq.dpd.shaping.table.set_interp(ipart_interpolation = enums.
↪IqOutEnvInterp.LINear)
```

No command help available

param ipart_interpolation
No help available

set_invert(*ipart_invert_values: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:[TABLe]:INVert
driver.source.iq.dpd.shaping.table.set_invert(ipart_invert_values = False)
```

No command help available

param ipart_invert_values
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.dpd.shaping.table.clone()
```

Subgroups

6.18.10.1.9.6 Amam

class AmamCls

Amam 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.iq.dpd.shaping.table.amam.clone()
```

Subgroups

6.18.10.1.9.7 File

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:CATalog
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:DATA
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:NEW
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:[SELect]
```

class FileCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:CATalog
value: List[str] = driver.source.iq.dpd.shaping.table.amam.file.get_catalog()
```

No command help available

return
catalog: No help available

get_data() → List[float]

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:DATA
value: List[float] = driver.source.iq.dpd.shaping.table.amam.file.get_data()
```

No command help available

return
emul_sgt_dpd_am_table_data: No help available

get_select() → str

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:[SELect]
value: str = driver.source.iq.dpd.shaping.table.amam.file.get_select()
```

No command help available

return
filename: No help available

set_data(*emul_sgt_dpd_am_table_data: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:DATA
driver.source.iq.dpd.shaping.table.amam.file.set_data(emul_sgt_dpd_am_table_
↪data = [1.1, 2.2, 3.3])
```

No command help available

param emul_sgt_dpd_am_table_data
No help available

set_new(*ipartd_pi_db_emul_sgt_dpd_am_table_data_new_file: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:NEW
driver.source.iq.dpd.shaping.table.amam.file.set_new(ipartd_pi_db_emul_sgt_dpd_
↪am_table_data_new_file = [1.1, 2.2, 3.3])
```

No command help available

param ipartd_pi_db_emul_sgt_dpd_am_table_data_new_file
No help available

set_select(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMAM:FILE:[SELect]
driver.source.iq.dpd.shaping.table.amam.file.set_select(filename = 'abc')
```

No command help available

param filename
No help available

6.18.10.1.9.8 AmPm

class AmPmCls

AmPm 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.iq.dpd.shaping.table.amPm.clone()
```

Subgroups

6.18.10.1.9.9 File

SCPI Commands :

```
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:CATalog
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:DATA
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:NEW
[SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:[SELect]
```

class FileCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:CATalog
value: List[str] = driver.source.iq.dpd.shaping.table.amPm.file.get_catalog()
```

No command help available

return
catalog: No help available

get_data() → List[float]

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:DATA
value: List[float] = driver.source.iq.dpd.shaping.table.amPm.file.get_data()
```

No command help available

return
emul_sgt_dpd_pm_table_data: No help available

get_select() → str

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:[SELect]
value: str = driver.source.iq.dpd.shaping.table.amPm.file.get_select()
```

No command help available

return
filename: No help available

set_data(*emul_sgt_dpd_pm_table_data: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:DATA
driver.source.iq.dpd.shaping.table.amPm.file.set_data(emul_sgt_dpd_pm_table_
↪data = [1.1, 2.2, 3.3])
```

No command help available

param emul_sgt_dpd_pm_table_data
No help available

set_new(*ipartd_pi_db_emul_sgt_dpd_pm_table_data_new_file: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:NEW
driver.source.iq.dpd.shaping.table.amPm.file.set_new(ipartd_pi_db_emul_sgt_dpd_
↪pm_table_data_new_file = [1.1, 2.2, 3.3])
```

No command help available

param ipartd_pi_db_emul_sgt_dpd_pm_table_data_new_file
No help available

set_select(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:DPD:SHAPing:TABLe:AMPM:FILE:[SELect]
driver.source.iq.dpd.shaping.table.amPm.file.set_select(filename = 'abc')
```

No command help available

param filename
No help available

6.18.10.2 Impairment

SCPI Command :

```
[SOURce<HW>]:IQ:IMPairment:[STATe]
```

class ImpairmentCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:[STATe]
value: bool = driver.source.iq.impairment.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:[STATe]
driver.source.iq.impairment.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.iq.impairment.clone()
```

Subgroups

6.18.10.2.1 IqRatio

SCPI Command :

```
[SOURce<HW>]:IQ:IMPairment:IQRatio:[MAGNitude]
```

class IqRatioCls

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

get_magnitude() → float

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:IQRatio:[MAGNitude]
value: float = driver.source.iq.impairment.iqRatio.get_magnitude()
```

No command help available

return
magnitude: No help available

set_magnitude(*magnitude: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:IQRatio:[MAGNitude]
driver.source.iq.impairment.iqRatio.set_magnitude(magnitude = 1.0)
```

No command help available

param magnitude
No help available

6.18.10.2.2 Leakage

SCPI Commands :

```
[SOURce<HW>]:IQ:IMPairment:LEAKage:I
[SOURce<HW>]:IQ:IMPairment:LEAKage:Q
```

class LeakageCls

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

get_icomponent() → float

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:LEAKage:I
value: float = driver.source.iq.impairment.leakage.get_icomponent()
```

No command help available

return
ipart: No help available

get_qcomponent() → float

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:LEAKage:Q
value: float = driver.source.iq.impairment.leakage.get_qcomponent()
```

No command help available

return
qpart: No help available

set_icomponent(*ipart: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:LEAKage:I
driver.source.iq.impairment.leakage.set_icomponent(ipart = 1.0)
```

No command help available

param ipart
No help available

set_qcomponent(*qpart: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:LEAKage:Q
driver.source.iq.impairment.leakage.set_qcomponent(qpart = 1.0)
```

No command help available

param qpart
No help available

6.18.10.2.3 Quadrature

SCPI Command :

```
[SOURce<HW>]:IQ:IMPairment:QUADrature:[ANGLe]
```

class QuadratureCls

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

get_angle() → float

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:QUADrature:[ANGLe]
value: float = driver.source.iq.impairment.quadrature.get_angle()
```

No command help available

return
angle: No help available

set_angle(*angle: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:QUADrature:[ANGLe]
driver.source.iq.impairment.quadrature.set_angle(angle = 1.0)
```

No command help available

param angle
No help available

6.18.10.3 Output

SCPI Command :

```
[SOURce<HW>]:IQ:OUTPut:LEVel
```

class OutputCls

Output commands group definition. 83 total commands, 2 Subgroups, 1 group commands

get_level() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:LEVel
value: float = driver.source.iq.output.get_level()
```

No command help available

return
level: No help available

set_level(*level: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:LEVel
driver.source.iq.output.set_level(level = 1.0)
```

No command help available

param level
No help available

Cloning the Group

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

Subgroups

6.18.10.3.1 Analog

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:MODE
[SOURce<HW>]:IQ:OUTPut:[ANALog]:PRESet
[SOURce<HW>]:IQ:OUTPut:[ANALog]:TYPE
```

class AnalogCls

Analog commands group definition. 59 total commands, 4 Subgroups, 3 group commands

get_mode() → IqOutMode

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:MODE
value: enums.IqOutMode = driver.source.iq.output.analog.get_mode()
```

No command help available

return
mode: No help available

get_type_py() → IqOutType

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:TYPE
value: enums.IqOutType = driver.source.iq.output.analog.get_type_py()
```

No command help available

return
type_py: No help available

preset() → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:PRESet
driver.source.iq.output.analog.preset()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:PRESet
driver.source.iq.output.analog.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsSmcv.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: IqOutMode*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:MODE
driver.source.iq.output.analog.set_mode(mode = enums.IqOutMode.FIXed)
```

No command help available

param mode
No help available

set_type_py(*type_py: IqOutType*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:TYPE
driver.source.iq.output.analog.set_type_py(type_py = enums.IqOutType.
↪DIFFerential)
```

No command help available

param type_py
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.analog.clone()
```

Subgroups

6.18.10.3.1.1 Bias

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:I
[SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:Q
```

class BiasCls

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

get_icomponent() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:I
value: float = driver.source.iq.output.analog.bias.get_icomponent()
```

No command help available

return
ipart: No help available

get_qcomponent() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:Q
value: float = driver.source.iq.output.analog.bias.get_qcomponent()
```

No command help available

return
qpart: No help available

set_icomponent(*ipart: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:I
driver.source.iq.output.analog.bias.set_icomponent(ipart = 1.0)
```

No command help available

param ipart
No help available

set_qcomponent(*qpart: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:Q
driver.source.iq.output.analog.bias.set_qcomponent(qpart = 1.0)
```

No command help available

param qpart
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.analog.bias.clone()
```

Subgroups

6.18.10.3.1.2 Coupling

SCPI Command :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:COUPling:[STATe]
```

class CouplingCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:COUPling:[STATe]
value: bool = driver.source.iq.output.analog.bias.coupling.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:BIAS:COUPling:[STATe]
driver.source.iq.output.analog.bias.coupling.set_state(state = False)
```

No command help available

param state
No help available

6.18.10.3.1.3 Envelope

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:ADAPtion
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:BIAS
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:BINPut
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:DELay
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:ETRak
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:FDPD
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:GAIN
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:OFFSet
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:RIN
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:STATe
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:TERMination
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VREF
```

class EnvelopeCls

Envelope commands group definition. 47 total commands, 7 Subgroups, 12 group commands

get_adaption() → IqOutEnvAdaption

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:ADAPtion
value: enums.IqOutEnvAdaption = driver.source.iq.output.analog.envelope.get_
↪adaption()
```

No command help available

return
adaption_mode: No help available

get_bias() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:BIAS
value: float = driver.source.iq.output.analog.envelope.get_bias()
```

No command help available

return
bias: No help available

get_binput() → bool

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:BINPut
value: bool = driver.source.iq.output.analog.envelope.get_binput()
```

No command help available

return
bipolar_input: No help available

get_delay() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:DELay
value: float = driver.source.iq.output.analog.envelope.get_delay()
```

No command help available

return
delay: No help available

get_etrak() → IqOutEnvEtRak

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:ETRak
value: enums.IqOutEnvEtRak = driver.source.iq.output.analog.envelope.get_etrak()
```

No command help available

return
etrak_ifc_type: No help available

get_fdpd() → bool

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:FDPD
value: bool = driver.source.iq.output.analog.envelope.get_fdpd()
```

No command help available

return
calc_from_dpd_stat: No help available

get_gain() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:GAIN
value: float = driver.source.iq.output.analog.envelope.get_gain()
```

No command help available

return
gain: No help available

get_offset() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:OFFSet
value: float = driver.source.iq.output.analog.envelope.get_offset()
```

No command help available

return
offset: No help available

get_rin() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:RIN
value: float = driver.source.iq.output.analog.envelope.get_rin()
```

No command help available

return
ipart_nput_resistance: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:STATe
value: bool = driver.source.iq.output.analog.envelope.get_state()
```

No command help available

return
state: No help available

get_termination() → IqOutEnvTerm

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:TERMination
value: enums.IqOutEnvTerm = driver.source.iq.output.analog.envelope.get_
↪termination()
```

No command help available

return
termination: No help available

get_vref() → IqOutEnvVrEf

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VREF
value: enums.IqOutEnvVrEf = driver.source.iq.output.analog.envelope.get_vref()
```

No command help available

return
voltage_reference: No help available

set_adaption(*adaption_mode: IqOutEnvAdaption*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:ADAPtion
driver.source.iq.output.analog.envelope.set_adaption(adaption_mode = enums.
↪IqOutEnvAdaption.AUTO)
```

No command help available

param adaption_mode
No help available

set_bias(*bias: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:BIAS
driver.source.iq.output.analog.envelope.set_bias(bias = 1.0)
```

No command help available

param bias
No help available

set_binput(*bipolar_input: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:BINPut
driver.source.iq.output.analog.envelope.set_binput(bipolar_input = False)
```

No command help available

param bipolar_input
No help available

set_delay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:DELay
driver.source.iq.output.analog.envelope.set_delay(delay = 1.0)
```

No command help available

param delay
No help available

set_etrak(*etrak_ifc_type: IqOutEnvEtRak*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:ETRak
driver.source.iq.output.analog.envelope.set_etrak(etrak_ifc_type = enums.
↪IqOutEnvEtRak.ET1V2)
```

No command help available

param etrak_ifc_type
No help available

set_fdpd(*calc_from_dpd_stat: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:FDPD
driver.source.iq.output.analog.envelope.set_fdpd(calc_from_dpd_stat = False)
```

No command help available

param calc_from_dpd_stat
No help available

set_gain(*gain: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:GAIN
driver.source.iq.output.analog.envelope.set_gain(gain = 1.0)
```

No command help available

param gain
No help available

set_offset(*offset: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:OFFSet
driver.source.iq.output.analog.envelope.set_offset(offset = 1.0)
```

No command help available

param offset
No help available

set_rin(*ipart_nput_resistance: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:RIN
driver.source.iq.output.analog.envelope.set_rin(ipart_nput_resistance = 1.0)
```

No command help available

param ipart_nput_resistance
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:STATe
driver.source.iq.output.analog.envelope.set_state(state = False)
```

No command help available

param state
No help available

set_termination(*termination: IqOutEnvTerm*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:TERMination
driver.source.iq.output.analog.envelope.set_termination(termination = enums.
↪IqOutEnvTerm.GROund)
```

No command help available

param termination
No help available

set_vref(*voltage_reference: IqOutEnvVrEf*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VREF
driver.source.iq.output.analog.envelope.set_vref(voltage_reference = enums.
↪IqOutEnvVrEf.VCC)
```

No command help available

param voltage_reference
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.analog.envelope.clone()
```

Subgroups

6.18.10.3.1.4 Emf

SCPI Command :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:EMF:[STATe]
```

class EmfCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:EMF:[STATe]
value: bool = driver.source.iq.output.analog.envelope.emf.get_state()
```

No command help available

return
emf_state: No help available

set_state(*emf_state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:EMF:[STATe]
driver.source.iq.output.analog.envelope.emf.set_state(emf_state = False)
```

No command help available

param emf_state
No help available

6.18.10.3.1.5 Pin

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:PIN:MAX
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:PIN:MIN
```

class PinCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:PIN:MAX
value: float = driver.source.iq.output.analog.envelope.pin.get_max()
```

No command help available

return
pin_max: No help available

get_min() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:PIN:MIN
value: float = driver.source.iq.output.analog.envelope.pin.get_min()
```

No command help available

return
pin_min: No help available

set_max(*pin_max: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:PIN:MAX
driver.source.iq.output.analog.envelope.pin.set_max(pin_max = 1.0)
```

No command help available

param pin_max
No help available

set_min(*pin_min: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:PIN:MIN
driver.source.iq.output.analog.envelope.pin.set_min(pin_min = 1.0)
```

No command help available

param pin_min
No help available

6.18.10.3.1.6 Power

SCPI Command :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:POWer:OFFSet
```

class PowerCls

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

get_offset() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:POWer:OFFSet
value: float = driver.source.iq.output.analog.envelope.power.get_offset()
```

No command help available

return
power_offset: No help available

6.18.10.3.1.7 Shaping

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:INTerp
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:MODE
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:SCALe
```

class ShapingCls

Shaping commands group definition. 23 total commands, 6 Subgroups, 3 group commands

get_interp() → IqOutEnvInterp

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:INTerp
value: enums.IqOutEnvInterp = driver.source.iq.output.analog.envelope.shaping.
↪get_interp()
```

No command help available

return
ipart_interpolation: No help available

get_mode() → IqOutEnvShapeMode

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:MODE
value: enums.IqOutEnvShapeMode = driver.source.iq.output.analog.envelope.
↪shaping.get_mode()
```

No command help available

return
shaping_mode: No help available

get_scale() → IqOutEnvScale

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:SCALe
value: enums.IqOutEnvScale = driver.source.iq.output.analog.envelope.shaping.
↪get_scale()
```

No command help available

return
scale: No help available

set_interp(*ipart_interpolation: IqOutEnvInterp*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:INTerp
driver.source.iq.output.analog.envelope.shaping.set_interp(ipart_interpolation␣
↪= enums.IqOutEnvInterp.LINear)
```

No command help available

param ipart_interpolation
No help available

set_mode(*shaping_mode: IqOutEnvShapeMode*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:MODE
driver.source.iq.output.analog.envelope.shaping.set_mode(shaping_mode = enums.
↪IqOutEnvShapeMode.DETRoughing)
```

No command help available

param shaping_mode
No help available

set_scale(*scale: IqOutEnvScale*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:SCALe
driver.source.iq.output.analog.envelope.shaping.set_scale(scale = enums.
↪IqOutEnvScale.POWer)
```

No command help available

param scale
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.analog.envelope.shaping.clone()
```

Subgroups

6.18.10.3.1.8 Clipping

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:CLIPping:FROM
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:CLIPping:TO
```

class ClippingCls

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

get_from_py() → int

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:CLIPping:FROM
value: int = driver.source.iq.output.analog.envelope.shaping.clipping.get_from_
↪py()
```

No command help available

return
clipping_from: No help available

get_to() → int

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:CLIPping:TO
value: int = driver.source.iq.output.analog.envelope.shaping.clipping.get_to()
```

No command help available

return
clipping_to: No help available

set_from_py(*clipping_from: int*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:CLIPping:FROM
driver.source.iq.output.analog.envelope.shaping.clipping.set_from_py(clipping_
↪from = 1)
```

No command help available

param clipping_from
No help available

set_to(*clipping_to: int*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:CLIPping:TO
driver.source.iq.output.analog.envelope.shaping.clipping.set_to(clipping_to = 1)
```

No command help available

param clipping_to
No help available

6.18.10.3.1.9 Coefficients

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients:CATalog
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients:LOAD
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients:STORe
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients
```

class CoefficientsCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients:CATalog
value: List[str] = driver.source.iq.output.analog.envelope.shaping.coefficients.
↪get_catalog()
```

No command help available

return
catalog: No help available

get_value() → List[float]

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients
value: List[float] = driver.source.iq.output.analog.envelope.shaping.
↪coefficients.get_value()
```

No command help available

return
ipartq_out_env_poly_coeffs: No help available

load(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients:LOAD
driver.source.iq.output.analog.envelope.shaping.coefficients.load(filename =
↪'abc')
```

No command help available

param filename
No help available

set_store(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients:STORe
driver.source.iq.output.analog.envelope.shaping.coefficients.set_store(filename␣
↪= 'abc')
```

No command help available

param filename
No help available

set_value(*ipartq_out_env_poly_coeffs: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:COEFficients
driver.source.iq.output.analog.envelope.shaping.coefficients.set_value(ipartq_
↪out_env_poly_coeffs = [1.1, 2.2, 3.3])
```

No command help available

param ipartq_out_env_poly_coeffs
No help available

6.18.10.3.1.10 Detroughing

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:COUPling
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:FACTor
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:FUNCtion
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:PEXPonent
```

class DetroughingCls

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

get_coupling() → bool

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:COUPling
value: bool = driver.source.iq.output.analog.envelope.shaping.detroughing.get_
↪coupling()
```

No command help available

return
coupling_state: No help available

get_factor() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:FACTor
value: float = driver.source.iq.output.analog.envelope.shaping.detroughing.get_
↪factor()
```

No command help available

return
detr_factor: No help available

get_function() → IqOutEnvDetrFunc

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:FUNCtion
value: enums.IqOutEnvDetrFunc = driver.source.iq.output.analog.envelope.shaping.
↪detroughing.get_function()
```

No command help available

return
detr_function: No help available

get_pexponent() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:PEXPonent
value: float = driver.source.iq.output.analog.envelope.shaping.detroughing.get_
↪pexponent()
```

No command help available

return
power_exponent: No help available

set_coupling(*coupling_state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:COUPling
driver.source.iq.output.analog.envelope.shaping.detroughing.set_
↪coupling(coupling_state = False)
```

No command help available

param coupling_state
No help available

set_factor(*detr_factor: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:FACTor
driver.source.iq.output.analog.envelope.shaping.detroughing.set_factor(detr_
↪factor = 1.0)
```

No command help available

param detr_factor
No help available

set_function(*detr_function: IqOutEnvDetrFunc*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:FUNCtion
driver.source.iq.output.analog.envelope.shaping.detroughing.set_function(detr_
↪function = enums.IqOutEnvDetrFunc.F1)
```

No command help available

param detr_function
No help available

set_pexponent(*power_exponent: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:DETRoughing:PEXPonent
driver.source.iq.output.analog.envelope.shaping.detroughing.set_pexponent(power_
↪exponent = 1.0)
```

No command help available

param power_exponent
No help available

6.18.10.3.1.11 File

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:CATalog
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:DATA
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:NEW
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:[SELect]
```

class FileCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:CATalog
value: List[str] = driver.source.iq.output.analog.envelope.shaping.file.get_
↪catalog()
```

No command help available

return
catalog: No help available

get_data() → List[float]

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:DATA
value: List[float] = driver.source.iq.output.analog.envelope.shaping.file.get_
↪data()
```

No command help available

return
emul_sgt_iq_out_env_shape_data: No help available

get_select() → str

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:[SELect]
value: str = driver.source.iq.output.analog.envelope.shaping.file.get_select()
```

No command help available

return
filename: No help available

set_data(*emul_sgt_iq_out_env_shape_data: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:DATA
driver.source.iq.output.analog.envelope.shaping.file.set_data(emul_sgt_iq_out_
↪env_shape_data = [1.1, 2.2, 3.3])
```

No command help available

param emul_sgt_iq_out_env_shape_data
No help available

set_new(*ipartd_pi_db_emul_sgt_iq_out_env_shape_data_new_file: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:NEW
driver.source.iq.output.analog.envelope.shaping.file.set_new(ipartd_pi_db_emul_
↪sgt_iq_out_env_shape_data_new_file = [1.1, 2.2, 3.3])
```

No command help available

param ipartd_pi_db_emul_sgt_iq_out_env_shape_data_new_file
No help available

set_select(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:FILE:[SELect]
driver.source.iq.output.analog.envelope.shaping.file.set_select(filename = 'abc
↪')
```

No command help available

param filename
No help available

6.18.10.3.1.12 Gain

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:GAIN:POST
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:GAIN:PRE
```

class GainCls

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

get_post() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:GAIN:POST
value: float = driver.source.iq.output.analog.envelope.shaping.gain.get_post()
```

No command help available

return
post_gain: No help available

get_pre() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:GAIN:PRE
value: float = driver.source.iq.output.analog.envelope.shaping.gain.get_pre()
```

No command help available

return
pre_gain: No help available

set_post(*post_gain: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:GAIN:POST
driver.source.iq.output.analog.envelope.shaping.gain.set_post(post_gain = 1.0)
```

No command help available

param post_gain
No help available

set_pre(*pre_gain: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:GAIN:PRE
driver.source.iq.output.analog.envelope.shaping.gain.set_pre(pre_gain = 1.0)
```

No command help available

param pre_gain
No help available

6.18.10.3.1.13 Pv

class PvCls

Pv 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.iq.output.analog.envelope.shaping.pv.clone()
```

Subgroups

6.18.10.3.1.14 File

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:CATalog
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:DATA
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:NEW
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:[SELect]
```

class FileCls

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

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:CATalog
value: List[str] = driver.source.iq.output.analog.envelope.shaping.pv.file.get_
↪catalog()
```

No command help available

return
catalog: No help available

get_data() → List[float]

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:DATA
value: List[float] = driver.source.iq.output.analog.envelope.shaping.pv.file.
↪get_data()
```

No command help available

return
emul_sgt_iq_out_env_shape_data_pv: No help available

get_select() → str

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:[SELect]
value: str = driver.source.iq.output.analog.envelope.shaping.pv.file.get_
↪select()
```

No command help available

return
filename: No help available

set_data(*emul_sgt_iq_out_env_shape_data_pv: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:DATA
driver.source.iq.output.analog.envelope.shaping.pv.file.set_data(emul_sgt_iq_
↪out_env_shape_data_pv = [1.1, 2.2, 3.3])
```

No command help available

param emul_sgt_iq_out_env_shape_data_pv
No help available

set_new(*ipartd_pi_db_emul_sgt_iq_out_env_shape_data_pv_new_file: List[float]*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:NEW
driver.source.iq.output.analog.envelope.shaping.pv.file.set_new(ipartd_pi_db_
↪emul_sgt_iq_out_env_shape_data_pv_new_file = [1.1, 2.2, 3.3])
```

No command help available

param ipartd_pi_db_emul_sgt_iq_out_env_shape_data_pv_new_file
No help available

set_select(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:SHAPing:PV:FILE:[SELect]
driver.source.iq.output.analog.envelope.shaping.pv.file.set_select(filename =
↪'abc')
```

No command help available

param filename
No help available

6.18.10.3.1.15 Vcc

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:MAX
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:MIN
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:OFFSet
```

class VccCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:MAX
value: float = driver.source.iq.output.analog.envelope.vcc.get_max()
```

No command help available

return
vcc_max: No help available

get_min() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:MIN
value: float = driver.source.iq.output.analog.envelope.vcc.get_min()
```

No command help available

return
vcc_min: No help available

get_offset() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:OFFSet
value: float = driver.source.iq.output.analog.envelope.vcc.get_offset()
```

No command help available

return
vcc_offset: No help available

set_max(*vcc_max: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:MAX
driver.source.iq.output.analog.envelope.vcc.set_max(vcc_max = 1.0)
```

No command help available

param vcc_max
No help available

set_min(*vcc_min: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:MIN
driver.source.iq.output.analog.envelope.vcc.set_min(vcc_min = 1.0)
```

No command help available

param vcc_min
No help available

`set_offset`(*vcc_offset: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:OFFSet
driver.source.iq.output.analog.envelope.vcc.set_offset(vcc_offset = 1.0)
```

No command help available

param vcc_offset
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.analog.envelope.vcc.clone()
```

Subgroups

6.18.10.3.1.16 Value

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:VALue:LEVel
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:VALue:PEP
```

`class ValueCls`

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

`get_level`() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:VALue:LEVel
value: float = driver.source.iq.output.analog.envelope.vcc.value.get_level()
```

No command help available

return
vcc_for_rf_level: No help available

`get_pep`() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VCC:VALue:PEP
value: float = driver.source.iq.output.analog.envelope.vcc.value.get_pep()
```

No command help available

return
vcc_for_crt_pep: No help available

6.18.10.3.1.17 Vout

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VOUT:MAX
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VOUT:MIN
```

class VoutCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VOUT:MAX
value: float = driver.source.iq.output.analog.envelope.vout.get_max()
```

No command help available

return
vout_max: No help available

get_min() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VOUT:MIN
value: float = driver.source.iq.output.analog.envelope.vout.get_min()
```

No command help available

return
vout_min: No help available

set_max(*vout_max: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VOUT:MAX
driver.source.iq.output.analog.envelope.vout.set_max(vout_max = 1.0)
```

No command help available

param vout_max
No help available

set_min(*vout_min: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VOUT:MIN
driver.source.iq.output.analog.envelope.vout.set_min(vout_min = 1.0)
```

No command help available

param vout_min
No help available

6.18.10.3.1.18 Vpp

SCPI Command :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VPP:[MAX]
```

class VppCls

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

get_max() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VPP:[MAX]
value: float = driver.source.iq.output.analog.envelope.vpp.get_max()
```

No command help available

return
vpp_max: No help available

set_max(*vpp_max: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:ENVelope:VPP:[MAX]
driver.source.iq.output.analog.envelope.vpp.set_max(vpp_max = 1.0)
```

No command help available

param vpp_max
No help available

6.18.10.3.1.19 Offset

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:OFFSet:I
[SOURce<HW>]:IQ:OUTPut:[ANALog]:OFFSet:Q
```

class OffsetCls

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

get_icomponent() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:OFFSet:I
value: float = driver.source.iq.output.analog.offset.get_icomponent()
```

No command help available

return
ipart: No help available

get_qcomponent() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:OFFSet:Q
value: float = driver.source.iq.output.analog.offset.get_qcomponent()
```

No command help available

return
qpart: No help available

set_icomponent(*ipart: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:OFFSet:I
driver.source.iq.output.analog.offset.set_icomponent(ipart = 1.0)
```

No command help available

param ipart
No help available

set_qcomponent(*qpart: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:OFFSet:Q
driver.source.iq.output.analog.offset.set_qcomponent(qpart = 1.0)
```

No command help available

param qpart
No help available

6.18.10.3.1.20 Setting

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:CATalog
[SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:DELete
[SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:LOAD
[SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:STORe
```

class SettingCls

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

delete(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:DELete
driver.source.iq.output.analog.setting.delete(filename = 'abc')
```

No command help available

param filename
No help available

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:CATalog
value: List[str] = driver.source.iq.output.analog.setting.get_catalog()
```

No command help available

return
catalog: No help available

load(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:LOAD
driver.source.iq.output.analog.setting.load(filename = 'abc')
```

No command help available

param filename
No help available

set_store(*filename: str*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:[ANALog]:SETTing:STORe
driver.source.iq.output.analog.setting.set_store(filename = 'abc')
```

No command help available

param filename
No help available

6.18.10.3.2 Digital

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:DIGital:CDEVice
[SOURce<HW>]:IQ:OUTPut:DIGital:INTerface
[SOURce<HW>]:IQ:OUTPut:DIGital:PON
[SOURce<HW>]:IQ:OUTPut:DIGital:STATe
```

class DigitalCls

Digital commands group definition. 23 total commands, 4 Subgroups, 4 group commands

get_cdevice() → str

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:CDEVice
value: str = driver.source.iq.output.digital.get_cdevice()
```

Queries information on the connected device.

return
cdevice: string

get_interface() → BbinInterfaceMode

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:INTerface
value: enums.BbinInterfaceMode = driver.source.iq.output.digital.get_interface()
```

Queries the connector at that the signal is output.

return
bbout_interf_mode: HSDin HSDin Dig. IQ HS x

get_pon() → UnchOff

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:PON
value: enums.UnchOff = driver.source.iq.output.digital.get_pon()
```

Sets the power-on state of the selected digital I/Q output.

return
pon: OFF| UNCHanged

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:STATe
value: bool = driver.source.iq.output.digital.get_state()
```

Activates the digital I/Q signal output.

return
state: 1| ON| 0| OFF

set_interface(*bbout_interf_mode: BbinInterfaceMode*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:INTerface
driver.source.iq.output.digital.set_interface(bbout_interf_mode = enums.
↪BbinInterfaceMode.DIGital)
```

Queries the connector at that the signal is output.

param bbout_interf_mode
HSDin HSDin Dig. IQ HS x

set_pon(*pon: UnchOff*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:PON
driver.source.iq.output.digital.set_pon(pon = enums.UnchOff.OFF)
```

Sets the power-on state of the selected digital I/Q output.

param pon
OFF| UNCHanged

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:STATe
driver.source.iq.output.digital.set_state(state = False)
```

Activates the digital I/Q signal output.

param state
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.digital.clone()
```

Subgroups

6.18.10.3.2.1 Channel<ChannelNull>

RepCap Settings

```
# Range: Nr0 .. Nr63
rc = driver.source.iq.output.digital.channel.repcap_channelNull_get()
driver.source.iq.output.digital.channel.repcap_channelNull_set(repcap.ChannelNull.Nr0)
```

class ChannelCls

Channel commands group definition. 5 total commands, 4 Subgroups, 0 group commands Repeated Capability: ChannelNull, default value after init: ChannelNull.Nr0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.digital.channel.clone()
```

Subgroups

6.18.10.3.2.2 Name

SCPI Command :

```
[SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:NAME
```

class NameCls

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

get(*channelNull=ChannelNull.Default*) → str

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:NAME
value: str = driver.source.iq.output.digital.channel.name.get(channelNull =␣
↪repcap.ChannelNull.Default)
```

Sets the channel name.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
dig_iq_hs_ch_name: string

set(*dig_iq_hs_ch_name: str*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:NAME
driver.source.iq.output.digital.channel.name.set(dig_iq_hs_ch_name = 'abc',␣
↪channelNull = repcap.ChannelNull.Default)
```

Sets the channel name.

param dig_iq_hs_ch_name
string

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.10.3.2.3 Power

class PowerCls

Power 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.iq.output.digital.channel.power.clone()
```

Subgroups

6.18.10.3.2.4 Level

SCPI Command :

```
[SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:POWer:LEVel
```

class LevelCls

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

get(*channelNull=ChannelNull.Default*) → float

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:POWer:LEVel
value: float = driver.source.iq.output.digital.channel.power.level.
↪get(channelNull = repcap.ChannelNull.Default)
```

No command help available

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbout_hs_level: No help available

6.18.10.3.2.5 Pep

SCPI Command :

```
[SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:POWer:PEP
```

class PepCls

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

get(*channelNull=ChannelNull.Default*) → float

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:POWer:PEP
value: float = driver.source.iq.output.digital.channel.power.pep.
↪get(channelNull = repcap.ChannelNull.Default)
```

No command help available

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
bbout_pep_hs: No help available

set(*bbout_pep_hs: float*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:POWer:PEP
driver.source.iq.output.digital.channel.power.pep.set(bbout_pep_hs = 1.0,␣
↪channelNull = repcap.ChannelNull.Default)
```

No command help available

param bbout_pep_hs
No help available

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.10.3.2.6 State

SCPI Command :

```
[SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:STATe
```

class StateCls

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

get(*channelNull=ChannelNull.Default*) → bool

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:STATe
value: bool = driver.source.iq.output.digital.channel.state.get(channelNull =␣
↪repcap.ChannelNull.Default)
```

Activates the channel.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
dig_iq_hs_out_ch_sta: 1| ON| 0| OFF

set(*dig_iq_hs_out_ch_sta: bool*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:STATe
driver.source.iq.output.digital.channel.state.set(dig_iq_hs_out_ch_sta = False,␣
↪channelNull = repcap.ChannelNull.Default)
```

Activates the channel.

param dig_iq_hs_out_ch_sta
1| ON| 0| OFF

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.10.3.2.7 SymbolRate

SCPI Command :

```
[SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:SRATe
```

class SymbolRateCls

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

get(*channelNull=ChannelNull.Default*) → float

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:SRATe
value: float = driver.source.iq.output.digital.channel.symbolRate.
↪get(channelNull = repcap.ChannelNull.Default)
```

Sets the sample rate of the channel of the HS digital I/Q output signal.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

return
dig_iq_hs_srat_chan: float Range: 400 to depends on options The maximum value depends on the connected receiving device. For more information, see data sheet.

set(*dig_iq_hs_srat_chan: float*, *channelNull=ChannelNull.Default*) → None

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:CHANnel<ST0>:SRATe
driver.source.iq.output.digital.channel.symbolRate.set(dig_iq_hs_srat_chan = 1.
↪0, channelNull = repcap.ChannelNull.Default)
```

Sets the sample rate of the channel of the HS digital I/Q output signal.

param dig_iq_hs_srat_chan
float Range: 400 to depends on options The maximum value depends on the connected receiving device. For more information, see data sheet.

param channelNull
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Channel')

6.18.10.3.2.8 Oflow

SCPI Command :

```
[SOURce]:IQ:OUTPut:DIGital:OFLow:STATe
```

class OflowCls

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

get_state() → bool

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:OFLow:STATe
value: bool = driver.source.iq.output.digital.oflow.get_state()
```

No command help available

return
state: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.digital.oflow.clone()
```

Subgroups

6.18.10.3.2.9 Hold

SCPI Commands :

```
[SOURce]:IQ:OUTPut:DIGital:OFLow:HOLD:RESet
[SOURce]:IQ:OUTPut:DIGital:OFLow:HOLD:STATe
```

class HoldCls

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

get_state() → bool

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:OFLow:HOLD:STATe
value: bool = driver.source.iq.output.digital.oflow.hold.get_state()
```

No command help available

return
state: No help available

reset() → None

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:OFLow:HOLD:RESet
driver.source.iq.output.digital.oflow.hold.reset()
```

No command help available

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

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:OFLow:HOLD:RESet
driver.source.iq.output.digital.oflow.hold.reset_with_opc()
```

No command help available

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

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

6.18.10.3.2.10 Power

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:DIGital:POWer:LEVel
[SOURce<HW>]:IQ:OUTPut:DIGital:POWer:PEP
[SOURce]:IQ:OUTPut:DIGital:POWer:VIA
```

class PowerCls

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

get_level() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:LEVel
value: float = driver.source.iq.output.digital.power.get_level()
```

No command help available

return
level: No help available

get_pep() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:PEP
value: float = driver.source.iq.output.digital.power.get_pep()
```

No command help available

return
pep: No help available

get_via() → IqOutDispViaType

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:POWer:VIA
value: enums.IqOutDispViaType = driver.source.iq.output.digital.power.get_via()
```

Selects the respective level entry field for the I/Q output.

return
via: PEP| LEVel

set_level(*level: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:LEVel
driver.source.iq.output.digital.power.set_level(level = 1.0)
```

No command help available

param level
No help available

set_pep(*pep: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:PEP
driver.source.iq.output.digital.power.set_pep(pep = 1.0)
```

No command help available

param pep
No help available

set_via(*via: IqOutDispViaType*) → None

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:POWer:VIA
driver.source.iq.output.digital.power.set_via(via = enums.IqOutDispViaType.
↪LEVel)
```

Selects the respective level entry field for the I/Q output.

param via
PEP| LEVel

Cloning the Group

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

Subgroups

6.18.10.3.2.11 Step

SCPI Commands :

```
[SOURce<HW>]:IQ:OUTPut:DIGital:POWer:STEP:MODE
[SOURce<HW>]:IQ:OUTPut:DIGital:POWer:STEP:[INCRement]
```

class StepCls

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

get_increment() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:STEP:[INCRement]
value: float = driver.source.iq.output.digital.power.step.get_increment()
```

Sets the step width. Use this value to vary the digital I/Q output level step-by-step.

return
ipart_increment: No help available

get_mode() → FreqStepMode

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:STEP:MODE
value: enums.FreqStepMode = driver.source.iq.output.digital.power.step.get_
↪mode()
```

Defines the type of step size to vary the digital output power step by step.

return
mode: DECimal| USER DECimal Increases or decreases the level in steps of 10 dB. USER Increases or decreases the level in increments, determined with the command [:SOURcehw]:IQ:OUTPut:DIGital:POWer:STEP[:INCRement].

set_increment(*ipart_increment: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:STEP:[INCRement]
driver.source.iq.output.digital.power.step.set_increment(ipart_increment = 1.0)
```

Sets the step width. Use this value to vary the digital I/Q output level step-by-step.

param ipart_increment
float Range: 0 to 100

set_mode(*mode: FreqStepMode*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:POWer:STEP:MODE
driver.source.iq.output.digital.power.step.set_mode(mode = enums.FreqStepMode.
↪DECimal)
```

Defines the type of step size to vary the digital output power step by step.

param mode
DECimal| USER DECimal Increases or decreases the level in steps of 10 dB. USER Increases or decreases the level in increments, determined with the command [:SOURcehw]:IQ:OUTPut:DIGital:POWer:STEP[:INCRement].

6.18.10.3.2.12 SymbolRate

SCPI Commands :

```
[SOURce]:IQ:OUTPut:DIGital:SRATe:MAX
[SOURce<HW>]:IQ:OUTPut:DIGital:SRATe:SOURce
[SOURce]:IQ:OUTPut:DIGital:SRATe:SUM
[SOURce<HW>]:IQ:OUTPut:DIGital:SRATe
```

class SymbolRateCls

SymbolRate commands group definition. 6 total commands, 2 Subgroups, 4 group commands

get_max() → int

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:SRATe:MAX
value: int = driver.source.iq.output.digital.symbolRate.get_max()
```

Queries the maximum supported sample rate.

return
dig_iqhs_in_sr_max: integer Range: 400 to depends on options The maximum value depends on the connected receiving device. For more information, see data sheet.

get_source() → BboutClocSour

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:SRATe:SOURce
value: enums.BboutClocSour = driver.source.iq.output.digital.symbolRate.get_
↪source()
```

No command help available

return
source: No help available

get_sum() → int

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:SRATe:SUM
value: int = driver.source.iq.output.digital.symbolRate.get_sum()
```

Queries the maximum supported sample rate.

return
dig_iqhs_in_sr_sum: integer Range: 400 to depends on options The maximum value depends on the connected receiving device. For more information, see data sheet.

get_value() → float

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:SRATe
value: float = driver.source.iq.output.digital.symbolRate.get_value()
```

Sets the sample rate of the digital I/Q output signal.

return
srate: float Range: 400 to depends on options, Unit: Hz The maximum value depends on the connected receiving device. For more information, see data sheet.

set_source(*source: BboutClocSour*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:SRATe:SOURce
driver.source.iq.output.digital.symbolRate.set_source(source = enums.
↪BboutClocSour.DIN)
```

No command help available

param source
No help available

set_value(*srate: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:SRATe
driver.source.iq.output.digital.symbolRate.set_value(srate = 1.0)
```

Sets the sample rate of the digital I/Q output signal.

param srate
float Range: 400 to depends on options, Unit: Hz The maximum value depends on the connected receiving device. For more information, see data sheet.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.iq.output.digital.symbolRate.clone()
```

Subgroups

6.18.10.3.2.13 Common

SCPI Command :

```
[SOURce]:IQ:OUTPut:DIGital:SRATe:COMMon:STATe
```

class CommonCls

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

get_state() → bool

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:SRATe:COMMon:STATe
value: bool = driver.source.iq.output.digital.symbolRate.common.get_state()
```

No command help available

return
dig_iq_hs_com_state: No help available

set_state(*dig_iq_hs_com_state: bool*) → None

```
# SCPI: [SOURce]:IQ:OUTPut:DIGital:SRATe:COMMon:STATe
driver.source.iq.output.digital.symbolRate.common.set_state(dig_iq_hs_com_state␣
↪= False)
```

No command help available

param dig_iq_hs_com_state
No help available

6.18.10.3.2.14 Fifo

SCPI Command :

```
[SOURce<HW>]:IQ:OUTPut:DIGital:SRATe:FIFO:[STATus]
```

class FifoCls

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

get_status() → SampRateFifoStatus

```
# SCPI: [SOURce<HW>]:IQ:OUTPut:DIGital:SRATe:FIFO:[STATus]
value: enums.SampRateFifoStatus = driver.source.iq.output.digital.symbolRate.
↪fifo.get_status()
```

No command help available

return
status: No help available

6.18.10.4 Swap

SCPI Command :

```
[SOURce<HW>]:IQ:SWAP:[STATe]
```

class SwapCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:SWAP:[STATe]
value: bool = driver.source.iq.swap.get_state()
```

Swaps the I and Q channel.

return
state: 1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:SWAP:[STATe]
driver.source.iq.swap.set_state(state = False)
```

Swaps the I and Q channel.

param state
1| ON| 0| OFF

6.18.11 Iqcoder

class IqcoderCls

Iqcoder commands group definition. 13 total commands, 8 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.18.11.1 Atsm

SCPI Command :

```
[SOURce]:[IQCoder]:ATSM:INPut
```

class AtsmCls

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

get_input_py() → CodingInputSignalInputSfe

```
# SCPI: [SOURce]:[IQCoder]:ATSM:INPut
value: enums.CodingInputSignalInputSfe = driver.source.iqcoder.atsm.get_input_
↪py()
```

No command help available

return
atscmh_input: No help available

set_input_py(*atscmh_input: CodingInputSignalInputSfe*) → None

```
# SCPI: [SOURce]:[IQCoder]:ATSM:INPut
driver.source.iqcoder.atsm.set_input_py(atscmh_input = enums.
↪CodingInputSignalInputSfe.ASI1)
```

No command help available

param atscmh_input
No help available

6.18.11.2 Dtmb

SCPI Command :

```
[SOURce]:[IQCoder]:DTMB:INPut
```

class DtmbCls

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

get_input_py() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:DTMB:INPut
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.dtmb.get_input_
↪py()
```

No command help available

return
dtmb_source: No help available

set_input_py(*dtmb_source: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:DTMB:INPut
driver.source.iqcoder.dtmb.set_input_py(dtmb_source = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param dtmb_source
No help available

6.18.11.3 Dvbc

SCPI Command :

```
[SOURce]:[IQCoder]:DVBC:INPut
```

class DvbcCls

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

get_input_py() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:DVBC:INPut
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.dvbc.get_input_
↪py()
```

No command help available

return
ipart_nput_sfe: No help available

set_input_py(*ipart_nput_sfe: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBC:INPut
driver.source.iqcoder.dvbc.set_input_py(ipart_nput_sfe = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param ipart_nput_sfe
No help available

6.18.11.4 Dvbs

SCPI Command :

```
[SOURce]:[IQCoder]:DVBS:INPut
```

class DvbsCls

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

get_input_py() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:DVBS:INPut
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.dvbs.get_input_
↪py()
```

No command help available

return
ipart_nput_sfe: No help available

set_input_py(*ipart_nput_sfe: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBS:INPut
driver.source.iqcoder.dvbs.set_input_py(ipart_nput_sfe = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param ipart_nput_sfe
No help available

6.18.11.5 Dvbs2

SCPI Commands :

```
[SOURce]:[IQCoder]:DVBS2:CONStel
[SOURce]:[IQCoder]:DVBS2:FECFrame
[SOURce]:[IQCoder]:DVBS2:INPut
[SOURce]:[IQCoder]:DVBS2:PILots
[SOURce]:[IQCoder]:DVBS2:RATE
```

class Dvbs2Cls

Dvbs2 commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_constel() → Dvbs2CodingConstelSfe

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:CONStel
value: enums.Dvbs2CodingConstelSfe = driver.source.iqcoder.dvbs2.get_constel()
```

No command help available

return
constel_sfe: No help available

get_fec_frame() → BicmFecFrame

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:FECFrame
value: enums.BicmFecFrame = driver.source.iqcoder.dvbs2.get_fec_frame()
```

No command help available

return
fec_frame_sfe: No help available

get_input_py() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:INPut
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.dvbs2.get_input_
↪py()
```

No command help available

return
ipart_nput_sfe: No help available

get_pilots() → bool

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:PILots
value: bool = driver.source.iqcoder.dvbs2.get_pilots()
```

No command help available

return
pilots_sfe: No help available

get_rate() → Dvbs2CodingCoderateSfe

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:RATE
value: enums.Dvbs2CodingCoderateSfe = driver.source.iqcoder.dvbs2.get_rate()
```

No command help available

return
rate_sfe: No help available

set_constel(*constel_sfe: Dvbs2CodingConstelSfe*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:CONStel
driver.source.iqcoder.dvbs2.set_constel(constel_sfe = enums.
↪Dvbs2CodingConstelSfe.A16)
```

No command help available

param constel_sfe
No help available

set_fec_frame(*fec_frame_sfe: BicmFecFrame*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:FECFrame
driver.source.iqcoder.dvbs2.set_fec_frame(fec_frame_sfe = enums.BicmFecFrame.
↪NORMal)
```

No command help available

param fec_frame_sfe
No help available

set_input_py(*ipart_nput_sfe: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:INPut
driver.source.iqcoder.dvbs2.set_input_py(ipart_nput_sfe = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param ipart_nput_sfe
No help available

set_pilots(*pilots_sfe: bool*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:PILots
driver.source.iqcoder.dvbs2.set_pilots(pilots_sfe = False)
```

No command help available

param pilots_sfe
No help available

set_rate(*rate_sfe: Dvbs2CodingCoderateSfe*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBS2:RATE
driver.source.iqcoder.dvbs2.set_rate(rate_sfe = enums.Dvbs2CodingCoderateSfe.R1_
↪2)
```

No command help available

param rate_sfe
No help available

6.18.11.6 Dvbt

class DvbtCls

Dvbt 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.source.iqcoder.dvbt.clone()
```

Subgroups

6.18.11.6.1 InputPy

SCPI Commands :

```
[SOURce]:[IQCoder]:DVBT:INPut:LOW
[SOURce]:[IQCoder]:DVBT:INPut:[HIGH]
```

class InputPyCls

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

get_high() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:DVBT:INPut:[HIGH]
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.dvbt.inputPy.get_
↪high()
```

No command help available

return
ipart_nput: No help available

get_low() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:DVBT:INPut:LOW
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.dvbt.inputPy.get_
↪low()
```

No command help available

return
ipart_nput_lp: No help available

set_high(*ipart_nput: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBT:INPut:[HIGH]
driver.source.iqcoder.dvbt.inputPy.set_high(ipart_nput = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param ipart_nput
No help available

set_low(*ipart_nput_lp: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:DVBT:INPut:LOW
driver.source.iqcoder.dvbt.inputPy.set_low(ipart_nput_lp = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param ipart_nput_lp
No help available

6.18.11.7 Isdbt

SCPI Command :

```
[SOURce]:[IQCoder]:ISDBt:INPut
```

class IsdbtCls

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

get_input_py() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:ISDBt:INPut
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.isdbt.get_input_
↪py()
```

No command help available

return
ipart_nput: No help available

set_input_py(*ipart_nput: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:ISDBt:INPut
driver.source.iqcoder.isdbt.set_input_py(ipart_nput = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param ipart_nput
No help available

6.18.11.8 J83B

SCPI Command :

```
[SOURce]:[IQCoder]:J83B:INPut
```

class J83BCls

J83B commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_input_py() → CodingInputSignalInputAsi

```
# SCPI: [SOURce]:[IQCoder]:J83B:INPut
value: enums.CodingInputSignalInputAsi = driver.source.iqcoder.j83B.get_input_
↪py()
```

No command help available

return
ipart_nput_sfe: No help available

set_input_py(*ipart_nput_sfe: CodingInputSignalInputAsi*) → None

```
# SCPI: [SOURce]:[IQCoder]:J83B:INPut
driver.source.iqcoder.j83B.set_input_py(ipart_nput_sfe = enums.
↪CodingInputSignalInputAsi.ASI1)
```

No command help available

param ipart_nput_sfe
No help available

6.18.12 ListPy

SCPI Commands :

```
[SOURce<HW>]:LIST:CATalog
[SOURce<HW>]:LIST:DELete
[SOURce<HW>]:LIST:DELete:ALL
[SOURce<HW>]:LIST:FREE
[SOURce<HW>]:LIST:MODE
[SOURce<HW>]:LIST:RESet
[SOURce<HW>]:LIST:RMODe
[SOURce<HW>]:LIST:RUNNing
[SOURce<HW>]:LIST:SELect
```

class ListPyCls

ListPy commands group definition. 32 total commands, 7 Subgroups, 9 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 RsSmcv.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_mode() → AutoStep

```
# SCPI: [SOURce<HW>]:LIST:MODE
value: enums.AutoStep = driver.source.listPy.get_mode()
```

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.

get_rmode() → LmodRunMode

```
# SCPI: [SOURce<HW>]:LIST:RMODe
value: enums.LmodRunMode = driver.source.listPy.get_rmode()
```

Selects the run mode for processing the list.

return
rmode: LEARned| LIVE LEARned Generates the signal by replaying the previously learned and saved data from the temporary memory. LIVE Generates the signal by processing the list directly.

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 RsSmcv.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: AutoStep*) → None

```
# SCPI: [SOURce<HW>]:LIST:MODE
driver.source.listPy.set_mode(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.

set_rmode(*rmode: LmodRunMode*) → None

```
# SCPI: [SOURce<HW>]:LIST:RMODe
driver.source.listPy.set_rmode(rmode = enums.LmodRunMode.LEARned)
```

Selects the run mode for processing the list.

param rmode
LEARned| LIVE LEARned Generates the signal by replaying the previously learned and saved data from the temporary memory. LIVE Generates the signal by processing the list directly.

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

6.18.12.1 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

6.18.12.1.1 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

6.18.12.1.1.1 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() → DexchSepDec

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:DECimal
value: enums.DexchSepDec = 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: DexchSepDec*) → None

```
# SCPI: [SOURce<HW>]:LIST:DEXChange:AFILe:SEParator:DECimal
driver.source.listPy.dexchange.afile.separator.set_decimal(decimal = enums.
↪DexchSepDec.COMMa)
```

Sets '.' (decimal point) or ',' (comma) as the decimal separator used in the ASCII data with floating-point numerals.

param decimal
DOT| COMMa

6.18.12.1.2 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.18.12.2 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: 0.5E-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: 0.5E-3 to 100

Cloning the Group

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

Subgroups

6.18.12.2.1 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 RsSmcv.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 RsSmcv.FormatPy.data for more details.

6.18.12.3 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 RsSmcv.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 RsSmcv.FormatPy.data. Range: 300 kHz to RFmax (depends on the installed options)

6.18.12.4 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

6.18.12.5 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()
```

Learns the selected list to determine the hardware setting for all list entries. The results are saved with the list. See also 'Learn List Mode Data list processing mode'.

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

```
# SCPI: [SOURce<HW>]:LIST:LEARn
driver.source.listPy.learn.set_with_opc()
```

Learns the selected list to determine the hardware setting for all list entries. The results are saved with the list. See also 'Learn List Mode Data list processing mode'.

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

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

6.18.12.6 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() → PowerAttMode

```
# SCPI: [SOURce<HW>]:LIST:POWer:AMODe
value: enums.PowerAttMode = 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 RsSmcv.FormatPy.data. Range: depends on the installed options , Unit: dBm

set_amode(*amode: PowerAttMode*) → None

```
# SCPI: [SOURce<HW>]:LIST:POWer:AMODe
driver.source.listPy.power.set_amode(amode = enums.PowerAttMode.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 RsSmcv.FormatPy.data. Range: depends on the installed options , Unit: dBm

6.18.12.7 Trigger

SCPI Command :

```
[SOURce<HW>]:LIST:TRIGger:SOURce
```

class TriggerCls

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

get_source() → TrigSweepSourNoHopExtAuto

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:SOURce
value: enums.TrigSweepSourNoHopExtAuto = driver.source.listPy.trigger.get_
↪source()
```

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_source(*source: TrigSweepSourNoHopExtAuto*) → None

```
# SCPI: [SOURce<HW>]:LIST:TRIGger:SOURce
driver.source.listPy.trigger.set_source(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.

Cloning the Group

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

Subgroups

6.18.12.7.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.18.13 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

6.18.13.1 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()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:MODulation:[ALL]:[STATe]
driver.source.modulation.all.set_state(state = False)
```

No command help available

param state
No help available

6.18.14 Noise

SCPI Command :

```
[SOURce]:NOISe:[STATe]
```

class NoiseCls

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

get_state() → NoisAwgnFseState

```
# SCPI: [SOURce]:NOISe:[STATe]
value: enums.NoisAwgnFseState = driver.source.noise.get_state()
```

No command help available

return
noise_state_mode: No help available

set_state(*noise_state_mode: NoisAwgnFseState*) → None

```
# SCPI: [SOURce]:NOISe:[STATe]
driver.source.noise.set_state(noise_state_mode = enums.NoisAwgnFseState.ADD)
```

No command help available

param noise_state_mode
No help available

Cloning the Group

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

Subgroups

6.18.14.1 Bandwidth

SCPI Command :

```
[SOURce<HW>]:NOISe:BWIDth:STATe
```

class BandwidthCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:NOISe:BWIDth:STATe
value: bool = driver.source.noise.bandwidth.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:NOISe:BWIDth:STATe
driver.source.noise.bandwidth.set_state(state = False)
```

No command help available

param state
No help available

6.18.14.2 Level

SCPI Command :

```
[SOURce<HW>]:NOISe:LEVel:RELative
```

class LevelCls

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

get_relative() → float

```
# SCPI: [SOURce<HW>]:NOISe:LEVel:RELative
value: float = driver.source.noise.level.get_relative()
```

No command help available

return
relative: No help available

6.18.15 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

6.18.16 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

6.18.16.1 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()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:PHASe:REFerence
driver.source.phase.reference.set_with_opc()
```

No command help available

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

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

6.18.17 Pm

SCPI Commands :

```
[SOURce<HW>]:PM:SENSitivity
[SOURce<HW>]:PM:[DEViation]
```

class PmCls

Pm commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_deviation() → float

```
# SCPI: [SOURce<HW>]:PM:[DEViation]
value: float = driver.source.pm.get_deviation()
```

No command help available

return
deviation: No help available

get_sensitivity() → float

```
# SCPI: [SOURce<HW>]:PM:SENSitivity
value: float = driver.source.pm.get_sensitivity()
```

No command help available

return
sensitivity: No help available

set_deviation(*deviation: float*) → None

```
# SCPI: [SOURce<HW>]:PM:[DEViation]
driver.source.pm.set_deviation(deviation = 1.0)
```

No command help available

param deviation
No help available

Cloning the Group

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

Subgroups

6.18.17.1 External

SCPI Commands :

```
[SOURce<HW>]:PM:EXTernal:COUPling
[SOURce<HW>]:PM:EXTernal:DEViation
```

class ExternalCls

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

get_coupling() → AcDc

```
# SCPI: [SOURce<HW>]:PM:EXTernal:COUPling
value: enums.AcDc = driver.source.pm.external.get_coupling()
```

No command help available

return
coupling: No help available

get_deviation() → float

```
# SCPI: [SOURce<HW>]:PM:EXTernal:DEViation
value: float = driver.source.pm.external.get_deviation()
```

No command help available

return
deviation: No help available

set_coupling(*coupling: AcDc*) → None

```
# SCPI: [SOURce<HW>]:PM:EXTernal:COUPling
driver.source.pm.external.set_coupling(coupling = enums.AcDc.AC)
```

No command help available

param coupling
No help available

set_deviation(*deviation: float*) → None

```
# SCPI: [SOURce<HW>]:PM:EXTernal:DEViation
driver.source.pm.external.set_deviation(deviation = 1.0)
```

No command help available

param deviation
No help available

6.18.17.2 Internal

SCPI Command :

```
[SOURce<HW>]:PM:INTernal:SOURce
```

class InternalCls

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

get_source() → AmSourceInt

```
# SCPI: [SOURce<HW>]:PM:INTernal:SOURce
value: enums.AmSourceInt = driver.source.pm.internal.get_source()
```

No command help available

return
source: No help available

set_source(*source: AmSourceInt*) → None

```
# SCPI: [SOURce<HW>]:PM:INTernal:SOURce
driver.source.pm.internal.set_source(source = enums.AmSourceInt.LF1)
```

No command help available

param source
No help available

6.18.18 Power

SCPI Commands :

```
[SOURce<HW>]:POWer:IQPep
[SOURce<HW>]:POWer:LBEHaviour
[SOURce<HW>]:POWer:MANual
[SOURce<HW>]:POWer:MODE
[SOURce<HW>]:POWer:PEP
[SOURce<HW>]:POWer:POWer
[SOURce<HW>]:POWer:SCHaracteristic
[SOURce<HW>]:POWer:STARt
[SOURce<HW>]:POWer:STOP
[SOURce]:POWer:WIGNore
```

class PowerCls

Power commands group definition. 42 total commands, 9 Subgroups, 10 group commands

get_iq_pep() → float

```
# SCPI: [SOURce<HW>]:POWer:IQPep
value: float = driver.source.power.get_iq_pep()
```

No command help available

return
ipart_qpep: No help available

get_lbehaviour() → PowLevBehaviour

```
# SCPI: [SOURce<HW>]:POWer:LBEHaviour
value: enums.PowLevBehaviour = driver.source.power.get_lbehaviour()
```

Set the RF level behaviour.

return
behaviour: AUTO| UNINterrupted UNINterrupted Do not use the uninterrupted level settings in combination with the high-quality optimization mode (see [:SOURcehw]:BB:IMPairment:OPTimization:MODE)

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_pep() → float

```
# SCPI: [SOURce<HW>]:POWer:PEP
value: float = driver.source.power.get_pep()
```

Queries the PEP 'Peak Envelope Power) of digital modulation or digital standards at the RF output. This value corresponds to the level specification, displayed in the status bar (header) .

return
pep: float

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_scharacteristic() → EmulSgtPowLevBehaviour

```
# SCPI: [SOURce<HW>]:POWer:SCHaracteristic
value: enums.EmulSgtPowLevBehaviour = driver.source.power.get_scharacteristic()
```

No command help available

return
characteristic: No help available

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()
```

No command help available

return
state: No help available

set_lbehaviour(*behaviour: PowLevBehaviour*) → None

```
# SCPI: [SOURce<HW>]:POWer:LBEHaviour
driver.source.power.set_lbehaviour(behaviour = enums.PowLevBehaviour.AUTO)
```

Set the RF level behaviour.

param behaviour
AUTO| UNINterrupted UNINterrupted Do not use the uninterrupted level settings in combination with the high-quality optimization mode (see [:SOURcehw]:BB:IMPairment:OPTimization:MODE)

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_scharacteristic(*characteristic: EmulSgtPowLevBehaviour*) → None

```
# SCPI: [SOURce<HW>]:POWer:SCHaracteristic
driver.source.power.set_scharacteristic(characteristic = enums.
↪EmulSgtPowLevBehaviour.AUTO)
```

No command help available

param characteristic
No help available

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)
```

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.power.clone()
```

Subgroups

6.18.18.1 Alc

SCPI Commands :

```
[SOURce<HW>]:POWer:ALC:DSENsitivity
[SOURce<HW>]:POWer:ALC:[STATe]
```

class AlcCls

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

get_dsensitivity() → PowAlcDetSensitivityEmulSgt

```
# SCPI: [SOURce<HW>]:POWer:ALC:DSENsitivity
value: enums.PowAlcDetSensitivityEmulSgt = driver.source.power.alc.get_
↪dsensitivity()
```

No command help available

return
sensitivity: No help available

get_state() → PowAlcStateEmulSgt

```
# SCPI: [SOURce<HW>]:POWer:ALC:[STATe]
value: enums.PowAlcStateEmulSgt = driver.source.power.alc.get_state()
```

No command help available

return
state: No help available

set_dsensitivity(*sensitivity: PowAlcDetSensitivityEmulSgt*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:DSENsitivity
driver.source.power.alc.set_dsensitivity(sensitivity = enums.
↪PowAlcDetSensitivityEmulSgt.AUTO)
```

No command help available

param sensitivity
No help available

set_state(*state: PowAlcStateEmulSgt*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:[STATe]
driver.source.power.alc.set_state(state = enums.PowAlcStateEmulSgt._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.source.power.alc.clone()
```

Subgroups

6.18.18.1.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.18.18.2 Attenuation

SCPI Commands :

```
[SOURce<HW>]:POWer:ATTenuation:DIGital
[SOURce<HW>]:POWer:ATTenuation:STAGe
```

class AttenuationCls

Attenuation commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_digital() → float

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:DIGital
value: float = driver.source.power.attenuation.get_digital()
```

No command help available

return
att_digital: No help available

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_digital(*att_digital: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:DIGital
driver.source.power.attenuation.set_digital(att_digital = 1.0)
```

No command help available

param att_digital
No help available

Cloning the Group

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

Subgroups

6.18.18.2.1 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.

6.18.18.2.2 Sover

SCPI Command :

```
[SOURce<HW>]:POWer:ATTenuation:SOVer:[OFFSet]
```

class SoverCls

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

get_offset() → float

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:SOVer:[OFFSet]
value: float = driver.source.power.attenuation.sover.get_offset()
```

No command help available

return
offset: No help available

set_offset(*offset: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:SOVer:[OFFSet]
driver.source.power.attenuation.sover.set_offset(offset = 1.0)
```

No command help available

param offset
No help available

6.18.18.3 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

6.18.18.4 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

6.18.18.4.1 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 RsSmcv.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: -200 to 200 , 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 RsSmcv.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: -200 to 200 , 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

6.18.18.5 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

6.18.18.6 Range

SCPI Commands :

```
[SOURce<HW>]:POWer:RANGe:LOWer
[SOURce<HW>]:POWer:RANGe:UPPer
```

class RangeCls

Range commands group definition. 2 total commands, 0 Subgroups, 2 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_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

6.18.18.7 Servoing

SCPI Commands :

```
[SOURce<HW>]:POWer:SERVoing:SET
[SOURce<HW>]:POWer:SERVoing:TARGet
[SOURce<HW>]:POWer:SERVoing:TEST
[SOURce<HW>]:POWer:SERVoing:TOLerance
[SOURce<HW>]:POWer:SERVoing:TRACking
```

class ServoingCls

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

class SetStruct

Structure for reading output parameters. Fields:

- Target: float: No parameter help available
- Start: enums.Test: No parameter help available

get_set() → SetStruct

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:SET
value: SetStruct = driver.source.power.servoing.get_set()
```

No command help available

return
structure: for return value, see the help for SetStruct structure arguments.

get_target() → float

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TARGet
value: float = driver.source.power.servoing.get_target()
```

No command help available

return
target_level: No help available

get_test() → Test

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TEST
value: enums.Test = driver.source.power.servoing.get_test()
```

No command help available

return
start: No help available

get_tolerance() → float

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TOLerance
value: float = driver.source.power.servoing.get_tolerance()
```

No command help available

return
tolerance: No help available

get_tracking() → bool

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TRACking
value: bool = driver.source.power.servoing.get_tracking()
```

No command help available

return
state: No help available

set_target(*target_level: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TARGet
driver.source.power.servoing.set_target(target_level = 1.0)
```

No command help available

param target_level
No help available

set_tolerance(*tolerance: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TOLerance
driver.source.power.servoing.set_tolerance(tolerance = 1.0)
```

No command help available

param tolerance
No help available

set_tracking(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TRACking
driver.source.power.servoing.set_tracking(state = False)
```

No command help available

param state
No help available

6.18.18.8 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()
```

Defines the capture range of the power control system. Within the range: Target Level +/- Catch Range the power control locks and tries to achieve the target level. Readings outside the range are not considered.

return
pow_cntrl_crange: float Range: 0 to 50

get_delay() → int

```
# SCPI: [SOURce<HW>]:POWer:SPC:DELay
value: int = driver.source.power.spc.get_delay()
```

Sets a waiting time for the generator to adjust the output level. After the delay time has elapsed, the power sensor measures the next value.

return
pow_cntrl_delay: integer Range: 0 to 1000

get_mode() → SensorModeAll

```
# SCPI: [SOURce<HW>]:POWer:SPC:MODE
value: enums.SensorModeAll = driver.source.power.spc.get_mode()
```

Selects the measurement mode for the power sensor.

return
control_mode: AUTO| SINGle AUTO Measures the level values continuously. SINGle Executes one measurement, triggered by the command [:SOURcehw]:POWer:SPC:SINGle.

get_peak() → bool

```
# SCPI: [SOURce<HW>]:POWer:SPC:PEAK
value: bool = driver.source.power.spc.get_peak()
```

Activates power control by means of the peak power values, provided the power sensor supports this function.

return
pow_cntrl_peak: 1| ON| 0| OFF

get_select() → PowCntrlSelect

```
# SCPI: [SOURce<HW>]:POWer:SPC:SELect
value: enums.PowCntrlSelect = driver.source.power.spc.get_select()
```

Selects the power sensor used for power control.

return
pow_cntrl_select: SENS1| SENS2| SENS3| SENS4| SENSor1| SENSor2| SENSor3| SENSor4

get_state() → bool

```
# SCPI: [SOURce<HW>]:POWer:SPC:STATe
value: bool = driver.source.power.spc.get_state()
```

Starts power control using the selected sensor. The control loop periodically adjusts the output level of the signal generator. After switching off, the running loop is completed.

return
pow_cntrl_state: 1| ON| 0| OFF

get_target() → float

```
# SCPI: [SOURce<HW>]:POWer:SPC:TARGet
value: float = driver.source.power.spc.get_target()
```

Sets the target level required at the DUT. To define the unit of the power value, use command method RsSmcv.Unit.power.

return
pow_cntrl_target: float Range: -50 to 30

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)
```

Defines the capture range of the power control system. Within the range: Target Level +/- Catch Range the power control locks and tries to achieve the target level. Readings outside the range are not considered.

param pow_cntrl_crange
float Range: 0 to 50

set_delay(*pow_cntrl_delay: int*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:DELay
driver.source.power.spc.set_delay(pow_cntrl_delay = 1)
```

Sets a waiting time for the generator to adjust the output level. After the delay time has elapsed, the power sensor measures the next value.

param pow_cntrl_delay
integer Range: 0 to 1000

set_mode(*control_mode: SensorModeAll*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:MODE
driver.source.power.spc.set_mode(control_mode = enums.SensorModeAll.AUTO)
```

Selects the measurement mode for the power sensor.

param control_mode
AUTO| SINGle AUTO Measures the level values continuously. SINGle Executes one measurement, triggered by the command [:SOURcehw]:POWer:SPC:SINGle.

set_peak(*pow_cntrl_peak: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:PEAK
driver.source.power.spc.set_peak(pow_cntrl_peak = False)
```

Activates power control by means of the peak power values, provided the power sensor supports this function.

param pow_cntrl_peak
1| ON| 0| OFF

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)
```

Selects the power sensor used for power control.

param pow_cntrl_select
SENS1| SENS2| SENS3| SENS4| SENSor1| SENSor2| SENSor3| SENSor4

set_state(*pow_cntrl_state: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:STATe
driver.source.power.spc.set_state(pow_cntrl_state = False)
```

Starts power control using the selected sensor. The control loop periodically adjusts the output level of the signal generator. After switching off, the running loop is completed.

param pow_cntrl_state
1| ON| 0| OFF

set_target(*pow_cntrl_target: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:TARGet
driver.source.power.spc.set_target(pow_cntrl_target = 1.0)
```

Sets the target level required at the DUT. To define the unit of the power value, use command method RsSmcv.Unit.power.

param pow_cntrl_target
float Range: -50 to 30

Cloning the Group

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

Subgroups

6.18.18.8.1 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()
```

Sets the measured power value as reference level.

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

```
# SCPI: [SOURce<HW>]:POWer:SPC:MEASure
driver.source.power.spc.measure.set_with_opc()
```

Sets the measured power value as reference level.

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

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

6.18.18.8.2 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()
```

Triggers the power sensor to measure the power value once.

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

```
# SCPI: [SOURce<HW>]:POWer:SPC:SINGle
driver.source.power.spc.single.set_with_opc()
```

Triggers the power sensor to measure the power value once.

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

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

6.18.18.9 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.

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.

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].

6.18.19 Pulm

SCPI Commands :

```
[SOURce<HW>]:PULM:DELay
[SOURce<HW>]:PULM:POLarity
[SOURce<HW>]:PULM:SOURce
[SOURce<HW>]:PULM:STATe
```

class PulmCls

Pulm commands group definition. 10 total commands, 2 Subgroups, 4 group commands

get_delay() → float

```
# SCPI: [SOURce<HW>]:PULM:DELay
value: float = driver.source.pulm.get_delay()
```

No command help available

return
delay: No help available

get_polarity() → NormalInverted

```
# SCPI: [SOURce<HW>]:PULM:POLarity
value: enums.NormalInverted = driver.source.pulm.get_polarity()
```

No command help available

return
polarity: No help available

get_source() → PulseSoure

```
# SCPI: [SOURce<HW>]:PULM:SOURce
value: enums.PulseSoure = driver.source.pulm.get_source()
```

No command help available

return
source: No help available

get_state() → bool

```
# SCPI: [SOURce<HW>]:PULM:STATe
value: bool = driver.source.pulm.get_state()
```

No command help available

return
state: No help available

set_delay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:DELay
driver.source.pulm.set_delay(delay = 1.0)
```

No command help available

param delay
No help available

set_polarity(*polarity: NormalInverted*) → None

```
# SCPI: [SOURce<HW>]:PULM:POLarity
driver.source.pulm.set_polarity(polarity = enums.NormalInverted.INVerted)
```

No command help available

param polarity
No help available

set_source(*source: PulseSoure*) → None

```
# SCPI: [SOURce<HW>]:PULM:SOURce
driver.source.pulm.set_source(source = enums.PulseSoure.CODer)
```

No command help available

param source
No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:PULM:STATe
driver.source.pulm.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.pulm.clone()
```

Subgroups

6.18.19.1 Double

SCPI Commands :

```
[SOURce<HW>]:PULM:DOUBle:STATe
[SOURce<HW>]:PULM:DOUBle:WIDTh
```

class DoubleCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:STATe
value: bool = driver.source.pulm.double.get_state()
```

No command help available

return
state: No help available

get_width() → float

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:WIDTh
value: float = driver.source.pulm.double.get_width()
```

No command help available

return
width: No help available

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:STATe
driver.source.pulm.double.set_state(state = False)
```

No command help available

param state
No help available

set_width(*width: float*) → None

```
# SCPI: [SOURce<HW>]:PULM:DOUBle:WIDTh
driver.source.pulm.double.set_width(width = 1.0)
```

No command help available

param width
No help available

6.18.19.2 Trigger

SCPI Command :

```
[SOURce<HW>]:PULM:TRIGger:MODE
```

class TriggerCls

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

get_mode() → PulsTrigMode

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:MODE
value: enums.PulsTrigMode = driver.source.pulm.trigger.get_mode()
```

No command help available

return
mode: No help available

set_mode(*mode: PulsTrigMode*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:MODE
driver.source.pulm.trigger.set_mode(mode = enums.PulsTrigMode.AUTO)
```

No command help available

param mode
No help available

Cloning the Group

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

Subgroups

6.18.19.2.1 External

SCPI Commands :

```
[SOURce<HW>]:PULM:TRIGger:EXTernal:IMPedance
[SOURce<HW>]:PULM:TRIGger:EXTernal:SLOPe
```

class ExternalCls

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

get_impedance() → InputImpRf

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:IMPedance
value: enums.InputImpRf = driver.source.pulm.trigger.external.get_impedance()
```

No command help available

return
impedance: No help available

get_slope() → SlopeType

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:SLOPe
value: enums.SlopeType = driver.source.pulm.trigger.external.get_slope()
```

No command help available

return
slope: No help available

set_impedance(*impedance: InputImpRf*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:IMPedance
driver.source.pulm.trigger.external.set_impedance(impedance = enums.InputImpRf.
↪G10K)
```

No command help available

param impedance
No help available

set_slope(*slope: SlopeType*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:SLOPe
driver.source.pulm.trigger.external.set_slope(slope = enums.SlopeType.NEGative)
```

No command help available

param slope
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.pulm.trigger.external.clone()
```

Subgroups

6.18.19.2.1.1 Gate

SCPI Command :

```
[SOURce<HW>]:PULM:TRIGger:EXTernal:GATE:POLarity
```

class GateCls

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

get_polarity() → NormalInverted

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:GATE:POLarity
value: enums.NormalInverted = driver.source.pulm.trigger.external.gate.get_
↪polarity()
```

No command help available

return
polarity: No help available

set_polarity(*polarity: NormalInverted*) → None

```
# SCPI: [SOURce<HW>]:PULM:TRIGger:EXTernal:GATE:POLarity
driver.source.pulm.trigger.external.gate.set_polarity(polarity = enums.
↪NormalInverted.INVerted)
```

No command help available

param polarity
No help available

6.18.20 Roscillator

SCPI Commands :

```
[SOURce]:ROSCillator:PRESet
[SOURce]:ROSCillator:SOURce
```

class RoscillatorCls

Roscillator commands group definition. 11 total commands, 3 Subgroups, 2 group commands

get_source() → RoscSourSetup

```
# SCPI: [SOURce]:ROSCillator:SOURce
value: enums.RoscSourSetup = 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 RsSmcv.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: RoscSourSetup*) → None

```
# SCPI: [SOURce]:ROSCillator:SOURce
driver.source.roscillator.set_source(source = enums.RoscSourSetup.ELOop)
```

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

6.18.20.1 External

SCPI Commands :

```
[SOURce]:ROSCillator:EXTernal:FREQuency
[SOURce]:ROSCillator:EXTernal:MLRange
[SOURce]:ROSCillator:EXTernal:NSBandwidth
[SOURce]:ROSCillator:EXTernal:SBANdwidth
```

class ExternalCls

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

get_frequency() → RoscFreqExt

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency
value: enums.RoscFreqExt = driver.source.roscillator.external.get_frequency()
```

Sets the frequency of the external reference.

return
frequency: 10MHZ| 13MHZ

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 synchronizsation 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_frequency(*frequency: RoscFreqExt*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency
driver.source.roscillator.external.set_frequency(frequency = enums.RoscFreqExt._
↪10MHZ)
```

Sets the frequency of the external reference.

param frequency
10MHZ| 13MHZ

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 synchronizsation 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

6.18.20.1.1 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

6.18.20.2 Internal

class InternalCls

Internal 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.source.roscillator.internal.clone()
```

Subgroups

6.18.20.2.1 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

6.18.20.3 Output

class OutputCls

Output 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.source.roscillator.output.clone()
```

Subgroups

6.18.20.3.1 Frequency

SCPI Commands :

```
[SOURce]:ROSCillator:OUTPut:FREQuency:MODE
[SOURce<HW>]:ROSCillator:OUTPut:FREQuency
```

class FrequencyCls

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

get_mode() → RoscOutpFreqMode

```
# SCPI: [SOURce]:ROSCillator:OUTPut:FREQuency:MODE
value: enums.RoscOutpFreqMode = driver.source.roscillator.output.frequency.get_
↪mode()
```

Selects the mode for the output reference frequency.

return
outp_freq_mode: DER10M| OFF| LOOPthrough OFF Disables the output. DER10M Sets the output reference frequency to 10 MHz. The reference frequency is derived from the internal reference frequency. LOOPthrough Forwards the input reference frequency to the reference frequency output.

get_value() → EmulSgtRoscOutputFreq

```
# SCPI: [SOURce<HW>]:ROSCillator:OUTPut:FREQuency
value: enums.EmulSgtRoscOutputFreq = driver.source.roscillator.output.frequency.
↪get_value()
```

No command help available

return
output_freq: No help available

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.DER10M)
```

Selects the mode for the output reference frequency.

param outp_freq_mode
DER10M| OFF| LOOPthrough OFF Disables the output. DER10M Sets the output reference frequency to 10 MHz. The reference frequency is derived from the internal reference frequency. LOOPthrough Forwards the input reference frequency to the reference frequency output.

set_value(*output_freq: EmulSgtRoscOutputFreq*) → None

```
# SCPI: [SOURce<HW>]:ROSCillator:OUTPut:FREQuency
driver.source.roscillator.output.frequency.set_value(output_freq = enums.
↪EmulSgtRoscOutputFreq._1000MHZ)
```

No command help available

param output_freq
No help available

6.18.21 Sweep

SCPI Command :

```
[SOURce<HW>]:SWEep:RESet:[ALL]
```

class SweepCls

Sweep commands group definition. 21 total commands, 2 Subgroups, 1 group commands

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 RsSmcv.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.sweep.clone()
```

Subgroups

6.18.21.1 Frequency

SCPI Commands :

```
[SOURce<HW>]:SWEep:[FREQuency]:DWELl
[SOURce<HW>]:SWEep:[FREQuency]:MODE
[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
```

class FrequencyCls

Frequency commands group definition. 10 total commands, 2 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: 1E-3 to 100, Unit: s

get_mode() → AutoManStep

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MODE
value: enums.AutoManStep = driver.source.sweep.frequency.get_mode()
```

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) .

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.

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: 1E-3 to 100, Unit: s

set_mode(*mode: AutoManStep*) → None

```
# SCPI: [SOURce<HW>]:SWEep:[FREQuency]:MODE
driver.source.sweep.frequency.set_mode(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) .

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.

Cloning the Group

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

Subgroups

6.18.21.1.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.18.21.1.2 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

6.18.21.2 Power

SCPI Commands :

```
[SOURce<HW>]:SWEep:POWer:AMODe
[SOURce<HW>]:SWEep:POWer:DWELl
[SOURce<HW>]:SWEep:POWer:MODE
[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. 10 total commands, 3 Subgroups, 7 group commands

get_amode() → PowerAttMode

```
# SCPI: [SOURce<HW>]:SWEep:POWer:AMODe
value: enums.PowerAttMode = 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: 10E-3 to 100, Unit: s

get_mode() → AutoManStep

```
# SCPI: [SOURce<HW>]:SWEep:POWer:MODE
value: enums.AutoManStep = driver.source.sweep.power.get_mode()
```

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].

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: PowerAttMode*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:AMODe
driver.source.sweep.power.set_amode(amode = enums.PowerAttMode.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: 10E-3 to 100, Unit: s

set_mode(*mode: AutoManStep*) → None

```
# SCPI: [SOURce<HW>]:SWEep:POWer:MODE
driver.source.sweep.power.set_mode(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].

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

6.18.21.2.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.18.21.2.2 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

6.18.21.2.3 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

6.19 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 RsSmcv.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 RsSmcv.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 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
```

(continues on next page)

(continued from previous page)

```
SYSTem:VERSion
SYSTem:WAIT
```

class SystemCls

System commands group definition. 197 total commands, 34 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 RsSmcv.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 RsSmcv.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')
```

Loads a file with previously saved R&S SMCV100B settings. Loads the selected file with previously saved R&S SMCV100B settings from the default or the specified directory. Loaded are files with extension *.savrcltxt.

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 SMCV100B settings into a file with defined filename and into a specified directory. The file extension (*.savrcltxt) is assigned 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 RsSmcv.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 RsSmcv.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

6.20.1 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

6.20.2 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

6.20.3 Communicate

class CommunicateCls

Communicate commands group definition. 33 total commands, 9 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.20.3.1 Bb

class BbCls

Bb 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.bb.clone()
```

Subgroups

6.20.3.1.1 Network

SCPI Command :

```
SYSTem:COMMunicate:BB<HW>:NETWork:PORT
```

class NetworkCls

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

get_port() → int

```
# SCPI: SYSTem:COMMunicate:BB<HW>:NETWork:PORT
value: int = driver.system.communicate.bb.network.get_port()
```

No command help available

return
port: No help available

set_port(*port: int*) → None

```
# SCPI: SYSTem:COMMunicate:BB<HW>:NETWork:PORT
driver.system.communicate.bb.network.set_port(port = 1)
```

No command help available

param port
No help available

6.20.3.2 BcIp

class BcIpCls

BcIp commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.20.3.2.1 Network

SCPI Commands :

```
SYSTem:COMMunicate:BCIP:NETWork:MACaddress
SYSTem:COMMunicate:BCIP:NETWork:PROTocol
SYSTem:COMMunicate:BCIP:NETWork:STATus
```

class NetworkCls

Network commands group definition. 8 total commands, 3 Subgroups, 3 group commands

get_mac_address() → str

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:MACaddress
value: str = driver.system.communicate.bcIp.network.get_mac_address()
```

Queries the MAC address of the network adapter.

return
mac_address: string Range: 00:00:00:00:00:00 to ff:ff:ff:ff:ff:ff

get_protocol() → NetProtocol

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:PROTocol
value: enums.NetProtocol = driver.system.communicate.bcIp.network.get_protocol()
```

Specifies the network protocol.

return
protocol: UDP

get_status() → bool

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:STATus
value: bool = driver.system.communicate.bcIp.network.get_status()
```

Queries the network connection state.

return
network_status: 1| ON| 0| OFF

set_mac_address(*mac_address: str*) → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:MACaddress
driver.system.communicate.bcIp.network.set_mac_address(mac_address = 'abc')
```

Queries the MAC address of the network adapter.

param mac_address
string Range: 00:00:00:00:00:00 to ff:ff:ff:ff:ff:ff

set_protocol(*protocol: NetProtocol*) → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:PROTocol
driver.system.communicate.bcIp.network.set_protocol(protocol = enums.
↪NetProtocol.TCP)
```

Specifies the network protocol.

param protocol
UDP

set_status(*network_status: bool*) → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:STATus
driver.system.communicate.bcIp.network.set_status(network_status = False)
```

Queries the network connection state.

param network_status
1| ON| 0| OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.communicate.bcIp.network.clone()
```

Subgroups

6.20.3.2.1.1 Common

SCPI Command :

```
SYSTem:COMMunicate:BCIP:NETWork:COMMon:HOSTname
```

class CommonCls

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

get_hostname() → str

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:COMMon:HOSTname
value: str = driver.system.communicate.bcIp.network.common.get_hostname()
```

Sets an individual hostname for the vector 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.

return
hostname: string

set_hostname(*hostname: str*) → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:COMMon:HOSTname
driver.system.communicate.bcIp.network.common.set_hostname(hostname = 'abc')
```

Sets an individual hostname for the vector 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.

param hostname
string

6.20.3.2.1.2 IpAddress

SCPI Commands :

```
SYSTem:COMMunicate:BCIP:NETWork:IPADdress:MODE
SYSTem:COMMunicate:BCIP:NETWork:IPADdress
```

class IpAddressCls

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

get_mode() → NetMode

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:IPADdress:MODE
value: enums.NetMode = driver.system.communicate.bcIp.network.ipAddress.get_
↪mode()
```

Selects manual or automatic setting of the IP address.

return
ip_mode: AUTO| STATic

get_value() → bytes

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:IPADdress
value: bytes = driver.system.communicate.bcIp.network.ipAddress.get_value()
```

Sets the IP address.

return
ip_net_ip_address: No help available

set_mode(*ip_mode: NetMode*) → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:IPADdress:MODE
driver.system.communicate.bcIp.network.ipAddress.set_mode(ip_mode = enums.
↪NetMode.AUTO)
```

Selects manual or automatic setting of the IP address.

param ip_mode
AUTO| STATic

set_value(*ip_net_ip_address: bytes*) → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:IPADdress
driver.system.communicate.bcIp.network.ipAddress.set_value(ip_net_ip_address = b
↪'ABCDEFGH')
```

Sets the IP address.

param ip_net_ip_address
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.communicate.bcIp.network.ipAddress.clone()
```

Subgroups

6.20.3.2.1.3 Subnet

SCPI Command :

```
SYSTem:COMMunicate:BCIP:NETWork:IPADdress:SUBNet:MASK
```

class SubnetCls

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

get_mask() → bytes

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:IPADdress:SUBNet:MASK
value: bytes = driver.system.communicate.bcIp.network.ipAddress.subnet.get_
↪mask()
```

Sets the subnet mask.

return
ip_net_sub_net_mask: No help available

set_mask(*ip_net_sub_net_mask: bytes*) → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:IPADdress:SUBNet:MASK
driver.system.communicate.bcIp.network.ipAddress.subnet.set_mask(ip_net_sub_net_
↪mask = b'ABCDEFGH')
```

Sets the subnet mask.

param ip_net_sub_net_mask
No help available

6.20.3.2.1.4 Restart

SCPI Command :

```
SYSTem:COMMunicate:BCIP:NETWork:RESTart
```

class RestartCls

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

set() → None

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:RESTart
driver.system.communicate.bcIp.network.restart.set()
```

Triggers a restart of the network.

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

```
# SCPI: SYSTem:COMMunicate:BCIP:NETWork:RESTart
driver.system.communicate.bcIp.network.restart.set_with_opc()
```

Triggers a restart of the network.

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

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

6.20.3.3 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()
```

No command help available

return
lterminator: No help available

get_resource() → str

```
# SCPI: SYSTem:COMMunicate:GPIB:RESource
value: str = driver.system.communicate.gpib.get_resource()
```

No command help available

return
resource: No help available

set_lterminator(*lterminator: IecTermMode*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB:LTERminator
driver.system.communicate.gpib.set_lterminator(lterminator = enums.IecTermMode.
↪EOI)
```

No command help available

param lterminator
No help available

Cloning the Group

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

Subgroups

6.20.3.3.1 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()
```

No command help available

return
address: No help available

set_address(*address: int*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB:[SELF]:ADDRess
driver.system.communicate.gpib.self.set_address(address = 1)
```

No command help available

param address
No help available

6.20.3.4 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

6.20.3.5 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 RsSmcv.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 RsSmcv.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

6.20.3.5.1 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 vector 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 RsSmcv.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 vector 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 RsSmcv.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

6.20.3.5.2 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

6.20.3.5.2.1 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

6.20.3.5.3 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.3.6 PciExpress

SCPI Command :

```
SYSTem:COMMunicate:PCIexpress:RESource
```

class PciExpressCls

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

get_resource() → str

```
# SCPI: SYSTem:COMMunicate:PCIexpress:RESource
value: str = driver.system.communicate.pciExpress.get_resource()
```

No command help available

return
resource: No help available

6.20.3.7 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

6.20.3.7.1 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

6.20.3.8 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()
```

No command help available

return
baud: No help available

get_parity() → Parity

```
# SCPI: SYSTem:COMMunicate:SERial:PARity
value: enums.Parity = driver.system.communicate.serial.get_parity()
```

No command help available

return
parity: No help available

get_resource() → str

```
# SCPI: SYSTem:COMMunicate:SERial:RESource
value: str = driver.system.communicate.serial.get_resource()
```

No command help available

return
resource: No help available

get_sbits() → Count

```
# SCPI: SYSTem:COMMunicate:SERial:SBITs
value: enums.Count = driver.system.communicate.serial.get_sbits()
```

No command help available

return
sbits: No help available

set_baud(*baud: Rs232BdRate*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:BAUD
driver.system.communicate.serial.set_baud(baud = enums.Rs232BdRate._115200)
```

No command help available

param baud
No help available

set_parity(*parity: Parity*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:PARity
driver.system.communicate.serial.set_parity(parity = enums.Parity.EVEN)
```

No command help available

param parity
No help available

`set_sbits`(*sbits: Count*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:SBITs
driver.system.communicate.serial.set_sbits(sbits = enums.Count._1)
```

No command help available

param sbits
No help available

6.20.3.9 Socket

SCPI Commands :

```
SYSTem:COMMunicate:SOCKet:PORT
SYSTem:COMMunicate:SOCKet:RESource
```

`class SocketCls`

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

`get_port`() → int

```
# SCPI: SYSTem:COMMunicate:SOCKet:PORT
value: int = driver.system.communicate.socket.get_port()
```

No command help available

return
scpi_eth_port: No help available

`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

`set_port`(*scpi_eth_port: int*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:PORT
driver.system.communicate.socket.set_port(scpi_eth_port = 1)
```

No command help available

param scpi_eth_port
No help available

6.20.4 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 RsSmcv.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 RsSmcv.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

6.20.5 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

6.20.6 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

6.20.6.1 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

6.20.7 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

6.20.7.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.7.2 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

6.20.7.2.1 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

6.20.7.2.2 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

6.20.7.3 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

6.20.8 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

6.20.8.1 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 RsSmcv.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 RsSmcv.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.

6.20.8.2 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 RsSmcv.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

6.20.9 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.10 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

6.20.11 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 RsSmcv.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

6.20.11.1 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

6.20.12 Identification

SCPI Commands :

```
SYSTem:IDENtification:PRESet
SYSTem:IDENtification
```

class IdentificationCls

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

get_value() → AutoUser

```
# SCPI: SYSTem:IDENtification
value: enums.AutoUser = driver.system.identification.get_value()
```

Selects the mode to determine the 'IDN String' and the 'OPT String' for the instrument, selected with command method RsSmcv.System.language. Note: While working in an emulation mode, the R&S SMCV100B specific command set is disabled, that is, the SCPI command method RsSmcv.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 RsSmcv.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: AutoUser*) → None

```
# SCPI: SYSTem:IDENtification
driver.system.identification.set_value(identification = enums.AutoUser.AUTO)
```

Selects the mode to determine the 'IDN String' and the 'OPT String' for the instrument, selected with command method RsSmcv.System.language. Note: While working in an emulation mode, the R&S SMCV100B specific command set is disabled, that is, the SCPI command method RsSmcv.System.Identification.value is discarded.

param identification
AUTO| USER AUTO Automatically determines the strings. USER User-defined strings can be selected.

6.20.13 Information

SCPI Commands :

```
SYSTem:INFormation:SR
SYSTem:INFormation
```

class InformationCls

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

get_value() → str

```
# SCPI: SYSTem:INFormation
value: str = driver.system.information.get_value()
```

No command help available

return
iec_idn: 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

6.20.14 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

6.20.14.1 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

6.20.15 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

6.20.15.1 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

6.20.15.2 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

6.20.15.3 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

6.20.15.4 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

6.20.15.4.1 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

6.20.15.5 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

6.20.16 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

6.20.16.1 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 SMCV100B stores user files on.

return
path_user: string

6.20.17 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

6.20.18 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

6.20.18.1 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

6.20.18.2 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

6.20.18.3 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

6.20.19 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

6.20.19.1 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 RsSmcv.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

6.20.19.1.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.19.1.2 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

6.20.19.1.3 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

6.20.20 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

6.20.20.1 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

6.20.20.2 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

6.20.20.3 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

6.20.20.4 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 RsSmcv.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

6.20.20.4.1 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

6.20.20.4.2 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

6.20.20.5 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

6.20.20.5.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.20.6 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

6.20.20.6.1 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

6.20.21 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

6.20.21.1 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')

6.20.22 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.23 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.24 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 RsSmcv.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

6.20.24.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.25 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

6.20.25.1 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

6.20.25.1.1 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

6.20.25.1.1.1 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

6.20.25.2 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

6.20.25.2.1 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

6.20.25.2.1.1 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()
```

No command help available

return
avahi_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param avahi_state
No help available

6.20.25.2.2 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

6.20.25.2.2.1 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()
```

No command help available

return
ftp_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param ftp_state
No help available

6.20.25.2.3 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

6.20.25.2.3.1 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()
```

No command help available

return
http_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param http_state
No help available

6.20.25.2.4 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

6.20.25.2.4.1 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()
```

No command help available

return
raw_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param raw_state
No help available

6.20.25.2.5 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()
```

No command help available

return
net_rem_support: No help available

set_state(*net_rem_support: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:REMSupport:[STATe]
driver.system.security.network.remSupport.set_state(net_rem_support = False)
```

No command help available

param net_rem_support
No help available

6.20.25.2.6 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

6.20.25.2.6.1 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()
```

No command help available

return
rpc_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param rpc_state
No help available

6.20.25.2.7 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

6.20.25.2.7.1 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()
```

No command help available

return
smb_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param smb_state
No help available

6.20.25.2.8 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

6.20.25.2.8.1 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()
```

No command help available

return
soe_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param soe_state
No help available

6.20.25.2.9 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

6.20.25.2.9.1 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()
```

No command help available

return
ssh_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param ssh_state
No help available

6.20.25.2.10 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()
```

No command help available

return
lan_stor_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param lan_stor_state
No help available

6.20.25.2.11 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

6.20.25.2.11.1 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()
```

No command help available

return
sw_update_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param sw_update_state
No help available

6.20.25.2.12 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

6.20.25.2.12.1 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()
```

No command help available

return
vnc_state: No help available

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)
```

No command help available

param sec_pass_word
No help available

param vnc_state
No help available

6.20.25.3 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

6.20.25.3.1 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

6.20.25.4 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

6.20.25.5 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

6.20.25.5.1 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

6.20.25.6 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

6.20.25.6.1 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

6.20.26 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.27 Specification

SCPI Commands :

```
SYSTem:SPECification:PARameter
SYSTem:SPECification
```

class SpecificationCls

Specification commands group definition. 7 total commands, 2 Subgroups, 2 group commands

get_parameter() → List[float]

```
# SCPI: SYSTem:SPECification:PARameter
value: List[float] = driver.system.specification.get_parameter()
```

Retrieves data sheet information for a specific parameter.

return
val_list: float Comma-separated list with the specified and, if available, the typical value of the parameter, as specified in the data sheet.

get_value() → List[float]

```
# SCPI: SYSTem:SPECification
value: List[float] = driver.system.specification.get_value()
```

Retrieves data sheet information for a specific parameter.

return
val_list: float Comma-separated list with the specified and, if available, the typical value of the parameter, as specified in the data sheet.

Cloning the Group

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

Subgroups

6.20.27.1 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)

6.20.27.2 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 RsSmcv.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 RsSmcv.System.Specification.Version.catalog.

param version
string

6.20.28 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 RsSmcv.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

6.20.29 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.30 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

6.20.30.1 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

6.20.31 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

6.20.32 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 RsSmcv.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 RsSmcv.System.Time.Zone.catalog and method RsSmcv.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 RsSmcv.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 RsSmcv.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 RsSmcv.System.Time.Zone.catalog and method RsSmcv.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 RsSmcv.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

6.20.32.1 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

6.20.32.1.1 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

6.20.32.2 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

6.20.32.2.1 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

6.20.32.3 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 RsSmcv.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 RsSmcv.System.Time.Zone.catalog.

param time_zone
string

6.20.33 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.

6.20.34 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

6.20.34.1 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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.20.34.2 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

6.20.34.3 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

6.21 Test

SCPI Commands :

```
TEST:EIQMode
TEST:LEVel
TEST:NRPTrigger
TEST:PRESet
```

class TestCls

Test commands group definition. 41 total commands, 13 Subgroups, 4 group commands

get_eiq_mode() → TestExtIqMode

```
# SCPI: TEST:EIQMode
value: enums.TestExtIqMode = driver.test.get_eiq_mode()
```

No command help available

return
eiq_mode: No help available

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 RsSmcv.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_eiq_mode(*eiq_mode: TestExtIqMode*) → None

```
# SCPI: TEST:EIQMode
driver.test.set_eiq_mode(eiq_mode = enums.TestExtIqMode.IQIN)
```

No command help available

param eiq_mode
No help available

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

6.21.1 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 RsSmcv.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 RsSmcv.utilities.opc_timeout_set() to set the timeout value.

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

6.21.2 Bb

SCPI Command :

```
TEST:BB:CONNection
```

class BbCls

Bb commands group definition. 13 total commands, 1 Subgroups, 1 group commands

get_connection() → bool

```
# SCPI: TEST:BB:CONNection
value: bool = driver.test.bb.get_connection()
```

No command help available

return
connection: No help available

Cloning the Group

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

Subgroups

6.21.2.1 Generator

SCPI Commands :

```
TEST:BB:GENerator:ARBitrary
TEST:BB:GENerator:SOURce
TEST:BB:GENerator:STATe
```

class GeneratorCls

Generator commands group definition. 12 total commands, 5 Subgroups, 3 group commands

get_arbitrary() → str

```
# SCPI: TEST:BB:GENerator:ARBitrary
value: str = driver.test.bb.generator.get_arbitrary()
```

No command help available

return
filename: No help available

get_source() → TestBbGenIqSour

```
# SCPI: TEST:BB:GENerator:SOURce
value: enums.TestBbGenIqSour = driver.test.bb.generator.get_source()
```

No command help available

return
iq_source: No help available

get_state() → bool

```
# SCPI: TEST:BB:GENerator:STATe
value: bool = driver.test.bb.generator.get_state()
```

No command help available

return
state: No help available

set_arbitrary(*filename: str*) → None

```
# SCPI: TEST:BB:GENerator:ARBitrary
driver.test.bb.generator.set_arbitrary(filename = 'abc')
```

No command help available

param filename
No help available

set_source(*iq_source: TestBbGenIqSour*) → None

```
# SCPI: TEST:BB:GENerator:SOURce
driver.test.bb.generator.set_source(iq_source = enums.TestBbGenIqSour.ARB)
```

No command help available

param iq_source
No help available

set_state(*state: bool*) → None

```
# SCPI: TEST:BB:GENerator:STATe
driver.test.bb.generator.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.test.bb.generator.clone()
```

Subgroups

6.21.2.1.1 Const

SCPI Commands :

```
TEST:BB:GENerator:CONSt:I
TEST:BB:GENerator:CONSt:Q
```

class ConstCls

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

get_icomponent() → float

```
# SCPI: TEST:BB:GENerator:CONSt:I
value: float = driver.test.bb.generator.const.get_icomponent()
```

No command help available

return
test_bb_gen_const_i: No help available

get_qcomponent() → float

```
# SCPI: TEST:BB:GENerator:CONSt:Q
value: float = driver.test.bb.generator.const.get_qcomponent()
```

No command help available

return
test_gen_const_q: No help available

set_icomponent(*test_bb_gen_const_i: float*) → None

```
# SCPI: TEST:BB:GENerator:CONSt:I
driver.test.bb.generator.const.set_icomponent(test_bb_gen_const_i = 1.0)
```

No command help available

param test_bb_gen_const_i
No help available

set_qcomponent(*test_gen_const_q: float*) → None

```
# SCPI: TEST:BB:GENerator:CONSt:Q
driver.test.bb.generator.const.set_qcomponent(test_gen_const_q = 1.0)
```

No command help available

param test_gen_const_q
No help available

6.21.2.1.2 Frequency<Index>

RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.test.bb.generator.frequency.repcap_index_get()
driver.test.bb.generator.frequency.repcap_index_set(repcap.Index.Nr1)
```

SCPI Command :

```
TEST:BB:GENerator:FREQuency<CH>
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Index, default value after init: Index.Nr1

get(*index=Index.Default*) → float

```
# SCPI: TEST:BB:GENerator:FREQuency<CH>
value: float = driver.test.bb.generator.frequency.get(index = repcap.Index.
↪Default)
```

No command help available

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
frequency: No help available

set(*frequency: float*, *index=Index.Default*) → None

```
# SCPI: TEST:BB:GENerator:FREQuency<CH>
driver.test.bb.generator.frequency.set(frequency = 1.0, index = repcap.Index.
↪Default)
```

No command help available

param frequency
No help available

param index
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

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

6.21.2.1.3 Gain

SCPI Commands :

```
TEST:BB:GENerator:GAIN:I
TEST:BB:GENerator:GAIN:Q
TEST:BB:GENerator:GAIN
```

class GainCls

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

get_icomponent() → float

```
# SCPI: TEST:BB:GENerator:GAIN:I
value: float = driver.test.bb.generator.gain.get_icomponent()
```

No command help available

return
test_gen_gain_i: No help available

get_qcomponent() → float

```
# SCPI: TEST:BB:GENerator:GAIN:Q
value: float = driver.test.bb.generator.gain.get_qcomponent()
```

No command help available

return
test_gen_gain_q: No help available

get_value() → float

```
# SCPI: TEST:BB:GENerator:GAIN
value: float = driver.test.bb.generator.gain.get_value()
```

No command help available

return
gain: No help available

set_icomponent(*test_gen_gain_i: float*) → None

```
# SCPI: TEST:BB:GENerator:GAIN:I
driver.test.bb.generator.gain.set_icomponent(test_gen_gain_i = 1.0)
```

No command help available

param test_gen_gain_i
No help available

set_qcomponent(*test_gen_gain_q: float*) → None

```
# SCPI: TEST:BB:GENerator:GAIN:Q
driver.test.bb.generator.gain.set_qcomponent(test_gen_gain_q = 1.0)
```

No command help available

param test_gen_gain_q
No help available

set_value(*gain: float*) → None

```
# SCPI: TEST:BB:GENerator:GAIN
driver.test.bb.generator.gain.set_value(gain = 1.0)
```

No command help available

param gain
No help available

6.21.2.1.4 Offset

SCPI Commands :

```
TEST:BB:GENerator:OFFSet:I
TEST:BB:GENerator:OFFSet:Q
```

class OffsetCls

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

get_icomponent() → float

```
# SCPI: TEST:BB:GENerator:OFFSet:I
value: float = driver.test.bb.generator.offset.get_icomponent()
```

No command help available

return
test_gen_offset_i: No help available

get_qcomponent() → float

```
# SCPI: TEST:BB:GENerator:OFFSet:Q
value: float = driver.test.bb.generator.offset.get_qcomponent()
```

No command help available

return
test_gen_offset_q: No help available

set_icomponent(*test_gen_offset_i: float*) → None

```
# SCPI: TEST:BB:GENerator:OFFSet:I
driver.test.bb.generator.offset.set_icomponent(test_gen_offset_i = 1.0)
```

No command help available

param test_gen_offset_i
No help available

set_qcomponent(*test_gen_offset_q: float*) → None

```
# SCPI: TEST:BB:GENerator:OFFSet:Q
driver.test.bb.generator.offset.set_qcomponent(test_gen_offset_q = 1.0)
```

No command help available

param test_gen_offset_q
No help available

6.21.2.1.5 Phase

SCPI Command :

```
TEST:BB:GENerator:PHASe:Q
```

class PhaseCls

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

get_qcomponent() → float

```
# SCPI: TEST:BB:GENerator:PHASe:Q
value: float = driver.test.bb.generator.phase.get_qcomponent()
```

No command help available

return
test_gen_phase_q: No help available

set_qcomponent(*test_gen_phase_q: float*) → None

```
# SCPI: TEST:BB:GENerator:PHASe:Q
driver.test.bb.generator.phase.set_qcomponent(test_gen_phase_q = 1.0)
```

No command help available

param test_gen_phase_q
No help available

6.21.3 Bbin

SCPI Commands :

```
TEST<HW>:BBIN:RBERror
TEST:BBIN
```

class BbinCls

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

get_rb_error() → bool

```
# SCPI: TEST<HW>:BBIN:RBERror
value: bool = driver.test.bbin.get_rb_error()
```

No command help available

return
rb_error: No help available

get_value() → bool

```
# SCPI: TEST:BBIN
value: bool = driver.test.bbin.get_value()
```

No command help available

return
bbin: No help available

6.21.4 BbOut

SCPI Command :

```
TEST<HW>:BBOut:LRATe
```

class BbOutCls

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

get_lrate() → float

```
# SCPI: TEST<HW>:BBOut:LRATe
value: float = driver.test.bbOut.get_lrate()
```

No command help available

return
lrate: No help available

set_lrate(*lrate: float*) → None

```
# SCPI: TEST<HW>:BBOut:LRATe
driver.test.bbOut.set_lrate(lrate = 1.0)
```

No command help available

param lrate
No help available

Cloning the Group

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

Subgroups

6.21.4.1 Ttest

SCPI Command :

```
TEST<HW>:BBOut:TTESt:[STATe]
```

class TtestCls

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

get_state() → bool

```
# SCPI: TEST<HW>:BBOut:TTESt:[STATe]
value: bool = driver.test.bbOut.ttest.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: TEST<HW>:BBOut:TTESt:[STATe]
driver.test.bbOut.ttest.set_state(state = False)
```

No command help available

param state
No help available

6.21.5 Connector

SCPI Commands :

```
TEST:CONNector:AUXio
TEST:CONNector:BNC
```

class ConnectorCls

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

get_aux_io() → bool

```
# SCPI: TEST:CONNector:AUXio
value: bool = driver.test.connector.get_aux_io()
```

No command help available

return
aux_io: No help available

get_bnc() → bool

```
# SCPI: TEST:CONNector:BNC
value: bool = driver.test.connector.get_bnc()
```

No command help available

return
bnc: No help available

6.21.6 Hs

SCPI Command :

```
TEST:HS
```

class HsCls

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

class GetStruct

Response structure. Fields:

- Interface: str: No parameter help available
- Result: str: No parameter help available

get(*get_py: str*) → GetStruct

```
# SCPI: TEST:HS
value: GetStruct = driver.test.hs.get(get_py = 'abc')
```

No command help available

param get_py
No help available

return
structure: for return value, see the help for GetStruct structure arguments.

set(*interface: str*, *set_py: str*) → None

```
# SCPI: TEST:HS
driver.test.hs.set(interface = 'abc', set_py = 'abc')
```

No command help available

param interface
No help available

param set_py
No help available

6.21.7 Keyboard

SCPI Command :

```
TEST:KEYBoard:[STATe]
```

class KeyboardCls

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

get_state() → bool

```
# SCPI: TEST:KEYBoard:[STATe]
value: bool = driver.test.keyboard.get_state()
```

No command help available

return
state: No help available

set_state(*state: bool*) → None

```
# SCPI: TEST:KEYBoard:[STATe]
driver.test.keyboard.set_state(state = False)
```

No command help available

param state
No help available

6.21.8 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

6.21.9 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

6.21.9.1 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

6.21.10 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

6.21.11 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

6.21.11.1 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

6.21.12 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

6.21.12.1 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

6.21.13 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

6.22 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. 6 total commands, 3 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

6.22.1 FreqSweep

class FreqSweepCls

FreqSweep 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.freqSweep.clone()
```

Subgroups

6.22.1.1 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
    - SWEep - all sweeps
    INTRO_CMD_HELP: Effective in the following configuration:

    - TRIG:FSW|PSW|[:SWE]:SOUR SING
    - SOUR:SWE:FREQ|POW: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

6.22.1.2 Source

SCPI Command :

```
TRIGger<HW>:FSWeep:SOURce
```

class SourceCls

Source commands group definition. 1 total commands, 0 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
    - 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 RsSmcv.Trigger.Sweep.Immediate.set, method RsSmcv.Trigger.Psweep.Immediate.set and method RsSmcv.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]: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)


    INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

    - FSWeep - RF 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 RsSmcv.Trigger.Sweep.Immediate.set, method RsSmcv.Trigger.Psweep.Immediate.set and method RsSmcv.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]: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')

6.22.2 Psweep

class PsweepCls

Psweep 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.psweep.clone()
```

Subgroups

6.22.2.1 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
    - SWEep - all sweeps
    INTRO_CMD_HELP: Effective in the following configuration:

    - TRIG:FSW|PSW|[:SWE]:SOUR SING
    - SOUR:SWE:FREQ|POW: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

6.22.2.2 Source

SCPI Command :

```
TRIGger<HW>:PSWeep:SOURce
```

class SourceCls

Source commands group definition. 1 total commands, 0 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)
```

(continues on next page)

(continued from previous page)

```
    INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

    - FSWeep - RF 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 RsSmcv.Trigger.Sweep.Immediate.set, method RsSmcv.Trigger.Psweep.Immediate.set and method RsSmcv.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]: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)


    INTRO_CMD_HELP: Selects the trigger source for the corresponding sweeps:

    - FSWeep - RF 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 RsSmcv.Trigger.Sweep.Immediate.set, method RsSmcv.Trigger.Psweep.Immediate.set and method RsSmcv.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]: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')

6.22.3 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

6.22.3.1 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
    - SWEep - all sweeps
    INTRO_CMD_HELP: Effective in the following configuration:

    - TRIG:FSW|PSW|[:SWE]:SOUR SING
    - SOUR:SWE:FREQ|POW: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

6.22.3.2 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
    - 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 RsSmcv.Trigger.Sweep.Immediate.set, method RsSmcv.Trigger.Psweep.Immediate.set and method RsSmcv.Trigger.FreqSweep.Immediate.set. Set the sweep mode with the commands: [:SOURcehw]:SWEep:POWer:MODEAUTO|STEP [:SOURcehw]: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')

6.23 TsGen

class TsGenCls

TsGen commands group definition. 21 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.23.1 Configure

SCPI Commands :

```
TSGen:CONFigure:COMMand
TSGen:CONFigure:PAYLoad
TSGen:CONFigure:PID
TSGen:CONFigure:PIDTestpack
TSGen:CONFigure:PLAYfile
TSGen:CONFigure:PLENgth
TSGen:CONFigure:STOPdata
TSGen:CONFigure:STUFfing
TSGen:CONFigure:TSPacket
TSGen:CONFigure:TSRate
```

class ConfigureCls

Configure commands group definition. 18 total commands, 3 Subgroups, 10 group commands

get_command() → TspLayerStatus

```
# SCPI: TSGen:CONFigure:COMMand
value: enums.TspLayerStatus = driver.tsGen.configure.get_command()
```

Triggers playing, pausing and stopping of the TS player file selected with method RsSmcv.TsGen.Configure.playFile.

return
player_status: STOP| PAUSe| PLAY| RESet

get_payload() → PayloadTestStuff

```
# SCPI: TSGen:CONFigure:PAYLoad
value: enums.PayloadTestStuff = driver.tsGen.configure.get_payload()
```

Determines the payload of the test packet. Also influences the payload of the generated stuffing packets while the TS player is running.

return
payload: HFF| H00| PRBS

get_pid() → int

```
# SCPI: TSGen:CONFigure:PID
value: int = driver.tsGen.configure.get_pid()
```

The available values depend on the settings of method RsSmcv.TsGen.Configure.pidTestPack. If method RsSmcv.TsGen. Configure.pidTestPack is set to NULL, then method RsSmcv.TsGen.Configure.pid is 1FFF(hex) . Otherwise the values are variable.

return
pid: integer Range: 0 to 8191

get_pid_test_pack() → PidTestPacket

```
# SCPI: TSGen:CONFigure:PIDTestpack
value: enums.PidTestPacket = driver.tsGen.configure.get_pid_test_pack()
```

Sets the PID, if method RsSmcv.TsGen.Configure.tsPacket is H184|H200|H204.

return
pid_test_pack: VARiable| NULL

get_play_file() → str

```
# SCPI: TSGen:CONFigure:PLAYfile
value: str = driver.tsGen.configure.get_play_file()
```

Specifies the file path and filename of the TS player file.

return
play_file: string

get_plength() → CodingPacketLength

```
# SCPI: TSGen:CONFigure:PLENgth
value: enums.CodingPacketLength = driver.tsGen.configure.get_plength()
```

Queries the packet length of the loaded file.

return
plength: P188| P204| P208| INV

get_stop_data() → All

```
# SCPI: TSGen:CONFigure:STOPdata
value: enums.All = driver.tsGen.configure.get_stop_data()
```

Ensures that a standardized TS data stream is always output at the TS output at the rear of the R&S SMCV100B.

return
stop_data: TTSP| NONE

get_stuffing() → bool

```
# SCPI: TSGen:CONFigure:STUFfing
value: bool = driver.tsGen.configure.get_stuffing()
```

Activates nullpacket stuffing.

return
stuffing: 1| ON| 0| OFF

get_ts_packet() → TspLayerSettingsTestTsPacket

```
# SCPI: TSGen:CONFigure:TSPacket
value: enums.TspLayerSettingsTestTsPacket = driver.tsGen.configure.get_ts_
↪packet()
```

Sets the structure of the generated test packets in pause or stop status.

return
ts_paket: H184| H200| H204| S187| S203| S207 S187|S203|S207 A sync byte (0x47) followed by 187/203/207 payload bytes. H184|H200|H204 A sync byte (0x47) followed by three header bytes and 184/200/204 payload bytes.

get_ts_rate() → int

```
# SCPI: TSGen:CONFigure:TSRate
value: int = driver.tsGen.configure.get_ts_rate()
```

Sets the output data rate of the player.

return
ts_rate: integer Range: 1 to 35E7

set_command(*player_status: TspLayerStatus*) → None

```
# SCPI: TSGen:CONFigure:COMMand
driver.tsGen.configure.set_command(player_status = enums.TspLayerStatus.PAUSe)
```

Triggers playing, pausing and stopping of the TS player file selected with method RsSmcv.TsGen.Configure.playFile.

param player_status
STOP| PAUSe| PLAY| RESet

set_payload(*payload: PayloadTestStuff*) → None

```
# SCPI: TSGen:CONFigure:PAYLoad
driver.tsGen.configure.set_payload(payload = enums.PayloadTestStuff.H00)
```

Determines the payload of the test packet. Also influences the payload of the generated stuffing packets while the TS player is running.

param payload
HFF| H00| PRBS

set_pid(*pid: int*) → None

```
# SCPI: TSGen:CONFigure:PID
driver.tsGen.configure.set_pid(pid = 1)
```

The available values depend on the settings of method RsSmcv.TsGen.Configure.pidTestPack. If method RsSmcv.TsGen. Configure.pidTestPack is set to NULL, then method RsSmcv.TsGen.Configure.pid is 1FFF(hex) . Otherwise the values are variable.

param pid
integer Range: 0 to 8191

set_pid_test_pack(*pid_test_pack: PidTestPacket*) → None

```
# SCPI: TSGen:CONFigure:PIDTestpack
driver.tsGen.configure.set_pid_test_pack(pid_test_pack = enums.PidTestPacket.
↪NULL)
```

Sets the PID, if method RsSmcv.TsGen.Configure.tsPacket is H184|H200|H204.

param pid_test_pack
VARiable| NULL

set_play_file(*play_file: str*) → None

```
# SCPI: TSGen:CONFigure:PLAYfile
driver.tsGen.configure.set_play_file(play_file = 'abc')
```

Specifies the file path and filename of the TS player file.

param play_file
string

set_plength(*plength: CodingPacketLength*) → None

```
# SCPI: TSGen:CONFigure:PLENgth
driver.tsGen.configure.set_plength(plength = enums.CodingPacketLength.INV)
```

Queries the packet length of the loaded file.

param plength
P188| P204| P208| INV

set_stop_data(*stop_data: All*) → None

```
# SCPI: TSGen:CONFigure:STOPdata
driver.tsGen.configure.set_stop_data(stop_data = enums.All.NONE)
```

Ensures that a standardized TS data stream is always output at the TS output at the rear of the R&S SMCV100B.

param stop_data
TTSP| NONE

set_stuffing(*stuffing: bool*) → None

```
# SCPI: TSGen:CONFigure:STUFfing
driver.tsGen.configure.set_stuffing(stuffing = False)
```

Activates nullpacket stuffing.

param stuffing
1| ON| 0| OFF

set_ts_packet(*ts_paket: TspLayerSettingsTestTsPacket*) → None

```
# SCPI: TSGen:CONFigure:TSPacket
driver.tsGen.configure.set_ts_packet(ts_paket = enums.
↪TspLayerSettingsTestTsPacket.H184)
```

Sets the structure of the generated test packets in pause or stop status.

param ts_paket
H184| H200| H204| S187| S203| S207 S187|S203|S207 A sync byte (0x47) followed by 187/203/207 payload bytes. H184|H200|H204 A sync byte (0x47) followed by three header bytes and 184/200/204 payload bytes.

set_ts_rate(*ts_rate: int*) → None

```
# SCPI: TSGen:CONFigure:TSRate
driver.tsGen.configure.set_ts_rate(ts_rate = 1)
```

Sets the output data rate of the player.

param ts_rate
integer Range: 1 to 35E7

Cloning the Group

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

Subgroups

6.23.1.1 Prbs

SCPI Command :

```
TSGen:CONFigure:PRBS:[SEQuence]
```

class PrbsCls

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

get_sequence() → SettingsPrbs

```
# SCPI: TSGen:CONFigure:PRBS:[SEQuence]
value: enums.SettingsPrbs = driver.tsGen.configure.prbs.get_sequence()
```

Sets the length of the PRBS sequence.

return
prbs: P15_1| P23_1

set_sequence(*prbs: SettingsPrbs*) → None

```
# SCPI: TSGen:CONFigure:PRBS:[SEQuence]
driver.tsGen.configure.prbs.set_sequence(prbs = enums.SettingsPrbs.P15_1)
```

Sets the length of the PRBS sequence.

param prbs
P15_1| P23_1

6.23.1.2 Seamless

SCPI Commands :

```
TSGen:CONFigure:SEAMless:CC
TSGen:CONFigure:SEAMless:PCR
TSGen:CONFigure:SEAMless:TT
```

class SeamlessCls

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

get_cc() → bool

```
# SCPI: TSGen:CONFigure:SEAMless:CC
value: bool = driver.tsGen.configure.seamless.get_cc()
```

Activates the correction of the continuity counters in the replayed TS data stream. The correction allows you to decode the stream without interruption when the play file is looping.

return
cc: 1| ON| 0| OFF

get_pcr() → bool

```
# SCPI: TSGen:CONFigure:SEAMless:PCR
value: bool = driver.tsGen.configure.seamless.get_pcr()
```

Activates the correction of time stamps in the replayed TS data stream. The correction allows you to decode the stream without interruption when the play file is looping.

return
pcr: 1| ON| 0| OFF

get_tt() → bool

```
# SCPI: TSGen:CONFigure:SEAMless:TT
value: bool = driver.tsGen.configure.seamless.get_tt()
```

Activates the correction of the time and date table in the replayed TS data stream. The correction allows you to decode the stream without interruption when the play file is looping.

return
tt: 1| ON| 0| OFF

set_cc(*cc: bool*) → None

```
# SCPI: TSGen:CONFigure:SEAMless:CC
driver.tsGen.configure.seamless.set_cc(cc = False)
```

Activates the correction of the continuity counters in the replayed TS data stream. The correction allows you to decode the stream without interruption when the play file is looping.

param cc
1| ON| 0| OFF

set_pcr(*pcr: bool*) → None

```
# SCPI: TSGen:CONFigure:SEAMless:PCR
driver.tsGen.configure.seamless.set_pcr(pcr = False)
```

Activates the correction of time stamps in the replayed TS data stream. The correction allows you to decode the stream without interruption when the play file is looping.

param pcr
1| ON| 0| OFF

set_tt(*tt: bool*) → None

```
# SCPI: TSGen:CONFigure:SEAMless:TT
driver.tsGen.configure.seamless.set_tt(tt = False)
```

Activates the correction of the time and date table in the replayed TS data stream. The correction allows you to decode the stream without interruption when the play file is looping.

param tt
1| ON| 0| OFF

6.23.1.3 Seek

SCPI Commands :

```
TSGen:CONFigure:SEEK:POSition
TSGen:CONFigure:SEEK:RESet
TSGen:CONFigure:SEEK:STARt
TSGen:CONFigure:SEEK:STOP
```

class SeekCls

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

get_position() → float

```
# SCPI: TSGen:CONFigure:SEEK:POSition
value: float = driver.tsGen.configure.seek.get_position()
```

Sets the position, that is the current playing time position. You can select a value in a 10-hour range.

return
position: float Range: 0 to 36000000

get_start() → float

```
# SCPI: TSGen:CONFigure:SEEK:STARt
value: float = driver.tsGen.configure.seek.get_start()
```

Sets an individual start time. You can select a value in a 10-hour range.

return
start: float Range: 0 to 36000000

get_stop() → float

```
# SCPI: TSGen:CONFigure:SEEK:STOP
value: float = driver.tsGen.configure.seek.get_stop()
```

Sets an individual stop time. You can select a value in a 10-hour range.

return
stop: float Range: 0 to 36000000

reset() → None

```
# SCPI: TSGen:CONFigure:SEEK:RESet
driver.tsGen.configure.seek.reset()


    INTRO_CMD_HELP: Resets the following parameters to their default state:

    - method RsSmcv.TsGen.Configure.Seek.start
    - method RsSmcv.TsGen.Configure.Seek.stop
```

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

```
# SCPI: TSGen:CONFigure:SEEK:RESet
driver.tsGen.configure.seek.reset_with_opc()


    INTRO_CMD_HELP: Resets the following parameters to their default state:

    - method RsSmcv.TsGen.Configure.Seek.start
    - method RsSmcv.TsGen.Configure.Seek.stop
```

Same as reset, but waits for the operation to complete before continuing further. Use the RsSmcv.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_position(*position: float*) → None

```
# SCPI: TSGen:CONFigure:SEEK:POSition
driver.tsGen.configure.seek.set_position(position = 1.0)
```

Sets the position, that is the current playing time position. You can select a value in a 10-hour range.

param position
float Range: 0 to 36000000

set_start(*start: float*) → None

```
# SCPI: TSGen:CONFigure:SEEK:STARt
driver.tsGen.configure.seek.set_start(start = 1.0)
```

Sets an individual start time. You can select a value in a 10-hour range.

param start
float Range: 0 to 36000000

set_stop(*stop: float*) → None

```
# SCPI: TSGen:CONFigure:SEEK:STOP
driver.tsGen.configure.seek.set_stop(stop = 1.0)
```

Sets an individual stop time. You can select a value in a 10-hour range.

param stop
float Range: 0 to 36000000

6.23.2 Read

SCPI Commands :

```
TSGen:READ:FMEMory
TSGen:READ:ORIGtsrate
```

class ReadCls

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

get_fmemory() → int

```
# SCPI: TSGen:READ:FMEMory
value: int = driver.tsGen.read.get_fmemory()
```

Queries the file size of the TS player file.

return
fmemory: integer Range: 0 to 10

get_orig_ts_rate() → int

```
# SCPI: TSGen:READ:ORIGtsrate
value: int = driver.tsGen.read.get_orig_ts_rate()
```

Displays the calculated original TS data rate.

return
orig_ts_rate: integer Range: 1 to 350000000

set_fmemory(*fmemory: int*) → None

```
# SCPI: TSGen:READ:FMEMory
driver.tsGen.read.set_fmemory(fmemory = 1)
```

Queries the file size of the TS player file.

param fmemory
integer Range: 0 to 10

set_orig_ts_rate(*orig_ts_rate: int*) → None

```
# SCPI: TSGen:READ:ORIGtsrate
driver.tsGen.read.set_orig_ts_rate(orig_ts_rate = 1)
```

Displays the calculated original TS data rate.

param orig_ts_rate
integer Range: 1 to 350000000

Cloning the Group

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

Subgroups

6.23.2.1 PlayFile

SCPI Command :

```
TSGen:READ:PLAYfile:LENGth
```

class PlayFileCls

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

get_length() → int

```
# SCPI: TSGen:READ:PLAYfile:LENGth
value: int = driver.tsGen.read.playFile.get_length()
```

Queries calculated original loop time.

return
length: integer Range: 0 to 100

6.24 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

RSSMCV UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsSmcv.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsSmcv.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 RsSmcv instance with new VISA session, the session gets a new thread lock. You can assign it to other RsSmcv sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsSmcv 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

RSSMCV 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

RSSMCV 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

T

U