RsSgt

Release 5.0.232.31

Rohde & Schwarz

Mar 25, 2024

CONTENTS:


SGT

CHAPTER

ONE

REVISION HISTORY

1.1 RsSgt

Rohde & Schwarz SGT100A SIGMA Vector Signal Generator RsSgt instrument driver.

Basic Hello-World code:

```
from RsSgt import *

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

Supported instruments: SGT100A

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

Documentation: https://RsSgt.readthedocs.io/

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

1.1.1 Version history:

Latest release notes summary: Update for FW 5.0.232, removed legacy option XM Radio

Version 5.0.232

- Update for FW 5.0.232
- Removed legacy option XM Radio

Version 4.90.109

- Updated for FW 4.90
- Updated core to the newest template.
- Added DigitalModulation Interface.

Version 4.80.1.25

- Fixed bug in interfaces with the name 'base'

Version 4.80.1.22

- Fixed several misspelled arguments and command headers

Version 4.80.1.19

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

Version 4.80.0.16

- Added arb_files interface
- Default HwInterface repcap is 0 (empty command suffix)

Version 4.80.0.5

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsSgt 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 RsSgt example:

```
"""Getting started - how to work with RsSgt Python package.
This example performs basic RF settings on an SGT100A instrument.
It shows the RsSgt calls and their corresponding SCPI commands.
Notice that the python RsSgt interfaces track the SCPI commands syntax."""

from RsSgt import *

# Open the session
sgt = RsSgt('TCPIP::10.112.0.228::HISLIP')
# Greetings, stranger...
print(f'Hello, I am: {sgt.utilities.idn_string}')

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

#   SOURce:POWer:LEVel:IMMediate:AMPLitude -25
sgt.source.power.level.immediate.set_amplitude(-25)
```

(continues on next page)

(continued from previous page)

```
#   SOURce:FREQuency:FIXed 1100000000
sgt.source.frequency.fixed.set_value(1.1E9)

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

# Close the session
sgt.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

RsSgt 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 RsSgt`

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 `RsSgt` 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 RsSgt offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsSgt needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsSgt package to your computer from the pypi.org: https://pypi.org/project/RsSgt/#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\RsSgt-5.0.232.31.tar`

2.3 Finding Available Instruments

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

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

from RsSgt import *

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

(continues on next page)

(continued from previous page)

```
instr_list = RsSgt.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 RsSgt 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 = RsSgt.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

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

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

- Installed RsSgt Python module Version 5.0.232 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsSgt import *

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

(continues on next page)

(continued from previous page)

```
RsSgt.assert_minimum_version('5.0.232')
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 = RsSgt(resource_string_1)

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

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

from RsSgt import *

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

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

from RsSgt import *

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

More `option_string` tokens are separated by comma:

```
driver = RsSgt('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 RsSgt objects:

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

from RsSgt import *

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

driver = RsSgt('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 RsSgt import *

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

driver = RsSgt('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

RsSgt 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 RsSgt 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 RsSgt import *

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

RsSgt 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 RsSgt 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 = RsSgt('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 RsSgt 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, RsSgt 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 RsSgt object
"""

import threading
from RsSgt import *


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


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

import threading
from RsSgt import *


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

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

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

(continues on next page)

(continued from previous page)

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

Run the following example:

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

import threading
from RsSgt import *


def execute(session: RsSgt, 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 = RsSgt('TCPIP::192.168.56.101::INSTR')
driver2 = RsSgt('TCPIP::192.168.56.101::INSTR')
driver1.utilities.visa_timeout = 200
driver2.utilities.visa_timeout = 200

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

threads = []
for i in range(10):
    t = threading.Thread(target=execute, args=(driver1, 1, i,))
    t.start()
    threads.append(t)
    t = threading.Thread(target=execute, args=(driver2, 2, i,))
    t.start()
```

(continues on next page)

(continued from previous page)

```
    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 RsSgt import *

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

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

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

Console output:

```
10:29:10.819     TCPIP::192.168.1.101::INSTR     0.976 ms  Write: *RST
10:29:10.819     TCPIP::192.168.1.101::INSTR  1884.985 ms  Status check: OK
10:29:12.704     TCPIP::192.168.1.101::INSTR     0.983 ms  Query OPC: 1
```

(continues on next page)

(continued from previous page)

```
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 = RsSgt('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 RsSgt import *

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
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 RsSgt import *

driver = RsSgt('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 AttMode

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

3.2 BbSystemConfiguration

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

3.3 CalPowDetAtt

```
# Example value:
value = enums.CalPowDetAtt.HIGH
# All values (4x):
HIGH | LOW | MED | OFF
```

3.4 FreqStepMode

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

3.5 ImpG50G10K

```
# Example value:
value = enums.ImpG50G10K.G10K
# All values (2x):
G10K | G50
```

3.6 InclExcl

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

3.7 InputImpRf

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

3.8 IqMode

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

3.9 OpMode

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

3.10 PowAlcState

```
# Example value:
value = enums.PowAlcState._1
# All values (4x):
_1 | AUTO | OFF | ON
```

3.11 PowLevBehaviour

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

3.12 PowLevMode

```
# Example value:
value = enums.PowLevMode.LOWDistortion
# All values (4x):
LOWDistortion | LOWNoise | NORMal | USER
```

3.13 PowOutpPonMode

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

3.14 PowRfOffMode

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

3.15 PowSensWithUndef

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

3.16 RefLoOutput

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

3.17 RoscFreqExt

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

3.18 SlopeType

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

3.19 SourceInt

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

3.20 Test

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

3.21 UserPlug

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

CHAPTER

FOUR

REPCAPS

4.1 HwInstance (Global)

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

4.2 BitNumberNull

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

4.3 IqConnector

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

4.4 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 RsSgt Python package.
This example performs basic RF settings on an SGT100A instrument.
It shows the RsSgt calls and their corresponding SCPI commands.
Notice that the python RsSgt interfaces track the SCPI commands syntax."""

from RsSgt import *

# Open the session
sgt = RsSgt('TCPIP::10.112.0.228::HISLIP')
# Greetings, stranger...
print(f'Hello, I am: {sgt.utilities.idn_string}')

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

#   SOURce:POWer:LEVel:IMMediate:AMPLitude -25
sgt.source.power.level.immediate.set_amplitude(-25)

#   SOURce:FREQuency:FIXed 1100000000
sgt.source.frequency.fixed.set_value(1.1E9)

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

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

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

from RsSgt import *


sgt = RsSgt('TCPIP::10.112.1.73::HISLIP')
# sgt = RsSgt('TCPIP::10.214.1.57::5025::SOCKET', options='SelectVisa=SocketIo') # No VISA␣
↪needed
print(f'Driver Info: {sgt.utilities.driver_version}')
```

(continues on next page)

(continued from previous page)

```
print(f'Instrument: {sgt.utilities.idn_string}')

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

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

sgt.output.state.set_value(True)
sgt.source.power.level.immediate.set_amplitude(-20)
sgt.source.frequency.fixed.set_value(223E6)
print(f'PEP level: {sgt.source.power.get_pep()} dBm')

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

```
"""The example:
 - creates waveform file from a csv-file with I/Q pairs
 - sends the file to the SGT100A instrument
 - activates the waveform
 You have the option of auto-scaling the samples to the full range with the parameter
↪'auto_scale'
"""

import numpy as np
from RsSgt import *

RsSgt.assert_minimum_version('4.70.1')
sgt = RsSgt('TCPIP::10.214.1.57::HISLIP')
print(sgt.utilities.idn_string)
sgt.utilities.reset()

pc_csv_file = r'c:\temp\arbFileExample.csv'
pc_wv_file = r'c:\temp\arbFileExample.wv'
instr_wv_file = '/var/user/InstrDemoFile.wv'

# Skip this part if you have a csv-file available

# Samples clock
clock_freq = 600e6
# wave clock
wave_freq = 120e6
# Scale factor - change it to less or more that 1 if you want to see the autoscaling␣
↪capability of the create_waveform_file...() methods
scale_factor = 0.43
time_vector = np.arange(0, 50 / wave_freq, 1 / clock_freq)
# I-component an Q-component data
i_data = np.cos(2 * np.pi * wave_freq * time_vector) * scale_factor
q_data = np.sin(2 * np.pi * wave_freq * time_vector) * scale_factor
```

(continues on next page)

(continued from previous page)

```
with open(pc_csv_file, 'w') as file:
    for x in range(len(i_data)):
        file.write(f'{i_data[x]},{q_data[x]}\n')

# Take that csv-file with the IQ-samples and create a wv file
result = sgt.arb_files.create_waveform_file_from_samples_file(pc_csv_file, pc_wv_file,␣
↪clock_freq=100E6, auto_scale=False, comment='Created from a csv file')
print(result)

# Send to the instrument
sgt.arb_files.send_waveform_file_to_instrument(pc_wv_file, instr_wv_file)

# Selecting the waveform and load it in ARB
sgt.source.bb.arbitrary.waveform.set_select(instr_wv_file)

# Turning on the ARB baseband
sgt.source.bb.arbitrary.set_state(True)

# Turning on the RF output state state
sgt.output.state.set_value(True)

sgt.close()
```

```
"""The example:
 - creates waveform file from two i_data and q_data vectors
 - sends the file to the SGT100A instrument
 - activates the waveform
 You have the option of auto-scaling the samples to the full range with the parameter
↪'auto_scale'
"""

import numpy as np
from RsSgt import *

RsSgt.assert_minimum_version('4.70.1')
sgt = RsSgt('TCPIP::10.214.1.57::HISLIP')
print(sgt.utilities.idn_string)
sgt.utilities.reset()

pc_wv_file = r'c:\temp\arbFileExample.wv'
instr_wv_file = '/var/user/InstrDemoFile.wv'

# Creating the I/Q vectors as lists: i_data / q_data
# Samples clock
clock_freq = 600e6
# wave clock
wave_freq = 120e6
# Scale factor - change it to less or more that 1 if you want to see the autoscaling␣
↪capability of the create_waveform_file...() methods
scale_factor = 0.8
time_vector = np.arange(0, 50 / wave_freq, 1 / clock_freq)
```

(continues on next page)

(continued from previous page)

```
# I-component an Q-component data
i_data = np.cos(2 * np.pi * wave_freq * time_vector) * scale_factor
q_data = np.sin(2 * np.pi * wave_freq * time_vector) * scale_factor

# Take those samples and create a wv file, send it to the instrument with the name instr_
↪wv_file (not auto-scaled)
result = sgt.arb_files.create_waveform_file_from_samples(i_data, q_data, pc_wv_file,␣
↪clock_freq=100E6, auto_scale=False, comment='Created from I/Q vectors')
sgt.arb_files.send_waveform_file_to_instrument(pc_wv_file, instr_wv_file)

# Selecting the waveform and load it in the ARB
sgt.source.bb.arbitrary.waveform.set_select(instr_wv_file)
sgt.source.frequency.fixed.set_value(1.1E9)
sgt.source.power.level.immediate.set_amplitude(-11.1)
# Turning on the ARB baseband
sgt.source.bb.arbitrary.set_state(True)
# Turning on the RF out state
sgt.output.state.set_value(True)

sgt.close()
```

```
"""Example showing how you can transfer a big file to the instrument and from the␣
↪instrument with showing the progress.
Since the SGT100A is quite fast on data transfer, we slow it down by waiting for 100ms␣
↪between each chunk transfer (1MB)
This way we see the transfer progress better and we do not need a file that is so big -␣
↪let's take cca 20MB.
For big files, use the example without the time.sleep(0.1)"""

import time
import numpy as np
from RsSgt import *


def my_transfer_handler(args):
    """Function called each time a chunk of data is transferred"""
    total_size = args.total_size if args.total_size is not None else "unknown"
    print(f"Context: '{args.context}{'with opc' if args.opc_sync else ''}', "
          f"chunk {args.chunk_ix}, "
          f"transferred {args.transferred_size} bytes, "
          f"total size {total_size}, "
          f"direction {'reading' if args.reading else 'writing'}, "
          f"data '{args.data}'")
    if args.end_of_transfer:
        print('End of Transfer')
    # Slow down the transfer by 200ms to see the progress better
    time.sleep(0.1)


sgt = RsSgt('TCPIP::10.214.1.57::HISLIP')
print(sgt.utilities.idn_string)
sgt.utilities.reset()
```

(continues on next page)

(continued from previous page)

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

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

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

CHAPTER

SIX

RSSGT API STRUCTURE

Global RepCaps

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

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

176 total commands, 10 Subgroups, 4 group commands

Initializes new RsSgt 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 RsSgt.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.

class LockStruct

Structure for reading output parameters. Fields:

- Lock_Request_Id: float: Number 0 test query to prove whether the instrument is locked Controller ID request lock from the controller with the specified Controller ID
- Value: float: Number 0 request refused; the instrument is already locked to other Lock Request Id, i.e. to another controller 1 request granted

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 RsSgt session.

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

Creates a new RsSgt 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()
```

Special command to set the RF output frequency with minimum latency. No unit (e.g. Hz) allowed. Bypasses the status system so command *OPC? cannot be appended.

return
freq: float

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() → LockStruct

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

Sends a lock request ID which uniquely identifies the controller to the instrument.

return
structure: for return value, see the help for LockStruct structure arguments.

get_pfast() → float

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

Special command to set the RF output level with minimum latency at the RF output connector. This value does not consider a specified offset. No unit (e.g. dBm) allowed. Bypasses the status system so command *OPC? cannot be appended.

return
power: float

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)
```

Special command to set the RF output frequency with minimum latency. No unit (e.g. Hz) allowed. Bypasses the status system so command *OPC? cannot be appended.

param freq
float

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)
```

Special command to set the RF output level with minimum latency at the RF output connector. This value does not consider a specified offset. No unit (e.g. dBm) allowed. Bypasses the status system so command *OPC? cannot be appended.

param power
float

unlock(*unlock_id: float*) → None

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

Unlocks an instrument locked to a controller with Controller ID = <Unlock Id>.

param unlock_id
Number Unlock ID which uniquely identifies the controller to the instrument. The value must match the Controller ID Lock Request Id set with the command [CMDLINKRESOLVED #Lock CMDLINKRESOLVED]. 0 Clear lock regardless of locking state

Subgroups

6.1 Calibration

class CalibrationCls

Calibration commands group definition. 10 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1 All

SCPI Command :

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

class AllCls

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

get_measure() → bool

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

Starts all internal adjustments for which no external measuring equipment is required.

return
all_py: 1| ON| 0| OFF

6.1.2 Frequency

SCPI Commands :

```
CALibration:FREQuency:TEMPerature
CALibration:FREQuency:[MEASure]
```

class FrequencyCls

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

get_measure() → bool

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

Performs all adjustments which affect the frequency.

return
synthesis: 1| ON| 0| OFF

get_temperature() → str

```
# SCPI: CALibration:FREQuency:TEMPerature
value: str = driver.calibration.frequency.get_temperature()
```

Queries the delta temperature since the last performed adjustment.

return
temperature: string

6.1.3 IqModulator

SCPI Commands :

```
CALibration:IQModulator:FULL
CALibration:IQModulator:LOCal
CALibration:IQModulator:TEMPerature
```

class IqModulatorCls

IqModulator commands group definition. 5 total commands, 2 Subgroups, 3 group commands

get_full() → bool

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

Starts the adjustment of the I/Q modulator for the entire frequency range. The I/Q modulator is adjusted with respect to carrier leakage, I/Q imbalance and quadrature.

return
modulator: 1| ON| 0| OFF

get_local() → bool

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

Starts the adjustment of the I/Q modulator for the current frequency. The I/Q modulator is adjusted with respect to carrier leakage, I/Q imbalance and quadrature. This adjustment is only possible when OUTPut[:STATe] ON and [:SOURce]:IQ:STATe ON.

return
cal_modulator_loc_error: No help available

get_temperature() → str

```
# SCPI: CALibration:IQModulator:TEMPerature
value: str = driver.calibration.iqModulator.get_temperature()
```

Queries the delta temperature since the last performed adjustment.

return
temperature: string

Cloning the Group

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

Subgroups

6.1.3.1 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: OFF| ON| 1| 0

set_state(*modulator: bool*) → None

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

No command help available

param modulator
OFF| ON| 1| 0

6.1.3.2 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()
```

Acitvates/deactivates a separat internal adjustment procedure for the I/Q modulator.

return
modulator: OFF| ON| 1| 0

set_state(*modulator: bool*) → None

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

Acitvates/deactivates a separat internal adjustment procedure for the I/Q modulator.

param modulator
OFF| ON| 1| 0

6.1.4 Level

SCPI Commands :

```
CALibration:LEVel:TEMPerature
CALibration:LEVel:[MEASure]
```

class LevelCls

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

get_measure() → bool

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

Starts all adjustments which affect the level.

return
level: 1| ON| 0| OFF

get_temperature() → str

```
# SCPI: CALibration:LEVel:TEMPerature
value: str = driver.calibration.level.get_temperature()
```

Queries the delta temperature since the last performed adjustment.

return
temperature: string

6.2 Connector

class ConnectorCls

Connector commands group definition. 7 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

6.2.1 RefLo

SCPI Command :

```
CONNector:REFLo:OUTPut
```

class RefLoCls

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

get_output() → RefLoOutput

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

Determines the signal provided at the output connector [REF/LO OUT] (rear of the instrument) .

return
output: REF| LO| OFF

set_output(*output: RefLoOutput*) → None

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

Determines the signal provided at the output connector [REF/LO OUT] (rear of the instrument) .

param output
REF| LO| OFF

6.2.2 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. 6 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.connector.user.clone()
```

Subgroups

6.2.2.1 Clock

class ClockCls

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

Subgroups

6.2.2.1.1 Impedance

SCPI Command :

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

class ImpedanceCls

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

get(*userIx=UserIx.Default*) → ImpG50G10K

```
# SCPI: CONNector:USER<CH>:CLOCk:IMPedance
value: enums.ImpG50G10K = driver.connector.user.clock.impedance.get(userIx =␣
↪repcap.UserIx.Default)
```

Selects the input impedance for the external trigger/clock inputs, when method RsSgt.Connector.User.Omode.set is set to TRIGger or CIN/COUT.

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

return
impedance: G50| G10K G10K Provided only for backward compatibility with other R&S signal generators. The R&S SGT accepts this values and maps it automatically to G1K.

set(*impedance: ImpG50G10K*, *userIx=UserIx.Default*) → None

```
# SCPI: CONNector:USER<CH>:CLOCk:IMPedance
driver.connector.user.clock.impedance.set(impedance = enums.ImpG50G10K.G10K,␣
↪userIx = repcap.UserIx.Default)
```

Selects the input impedance for the external trigger/clock inputs, when method RsSgt.Connector.User.Omode.set is set to TRIGger or CIN/COUT.

param impedance
G50| G10K G10K Provided only for backward compatibility with other R&S signal generators. The R&S SGT accepts this values and maps it automatically to G1K.

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

6.2.2.1.2 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)
```

Sets the polarity of the active slope of an applied instrument trigger/clock.

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

return
slope: NEGative| POSitive

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)
```

Sets the polarity of the active slope of an applied instrument trigger/clock.

param slope
NEGative| POSitive

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

6.2.2.2 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*) → UserPlug

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

Sets the operation mode of the user connector.

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

return
omode: MKR1| MKR2| TRIGger| CIN| COUT| SIN| SOUT| NEXT| LOW| MLATency| MARRived| HIGH| SVALid| SNValid| PVOut| PETRigger| PEMSource| TOUT MKR1/2 Marker 1/2 TRIGger Trigger TOUT Trigger out CIN Clock in COUT Clock out SIN Sync in SOUT Sync out NEXT Next trigger SVALid|SNValid Signal valid /not valid PVOut Pulse generator video out PETRigger Pulse generator external trigger PEMSource External pulse modulator source

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

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

Sets the operation mode of the user connector.

param omode
MKR1| MKR2| TRIGger| CIN| COUT| SIN| SOUT| NEXT| LOW| MLATency| MARRived| HIGH| SVALid| SNValid| PVOut| PETRigger| PEMSource| TOUT MKR1/2 Marker 1/2 TRIGger Trigger TOUT Trigger out CIN Clock in COUT Clock out SIN Sync in SOUT Sync out NEXT Next trigger SVALid|SNValid Signal valid /not valid PVOut Pulse generator video out PETRigger Pulse generator external trigger PEMSource External pulse modulator source

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

6.2.2.3 Threshold

SCPI Command :

```
CONNector:USER<CH>:THReshold
```

class ThresholdCls

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

get(*userIx=UserIx.Default*) → float

```
# SCPI: CONNector:USER<CH>:THReshold
value: float = driver.connector.user.threshold.get(userIx = repcap.UserIx.
↪Default)
```

Sets the threshold for the user connector.

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

return
threshold: float Range: 0 to 2

6.2.2.4 Trigger

class TriggerCls

Trigger 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.connector.user.trigger.clone()
```

Subgroups

6.2.2.4.1 Impedance

SCPI Command :

```
CONNector:USER<CH>:TRIGger:IMPedance
```

class ImpedanceCls

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

get(*userIx=UserIx.Default*) → ImpG50G10K

```
# SCPI: CONNector:USER<CH>:TRIGger:IMPedance
value: enums.ImpG50G10K = driver.connector.user.trigger.impedance.get(userIx =␣
↪repcap.UserIx.Default)
```

Selects the input impedance for the external trigger/clock inputs, when method RsSgt.Connector.User.Omode.set is set to TRIGger or CIN/COUT.

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

return
impedance: G50| G10K G10K Provided only for backward compatibility with other R&S signal generators. The R&S SGT accepts this values and maps it automatically to G1K.

set(*impedance: ImpG50G10K*, *userIx=UserIx.Default*) → None

```
# SCPI: CONNector:USER<CH>:TRIGger:IMPedance
driver.connector.user.trigger.impedance.set(impedance = enums.ImpG50G10K.G10K,␣
↪userIx = repcap.UserIx.Default)
```

Selects the input impedance for the external trigger/clock inputs, when method RsSgt.Connector.User.Omode.set is set to TRIGger or CIN/COUT.

param impedance
G50| G10K G10K Provided only for backward compatibility with other R&S signal generators. The R&S SGT accepts this values and maps it automatically to G1K.

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

6.2.2.4.2 Slope

SCPI Command :

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

class SlopeCls

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

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

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

Sets the polarity of the active slope of an applied instrument trigger/clock.

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

return
slope: NEGative| POSitive

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

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

Sets the polarity of the active slope of an applied instrument trigger/clock.

param slope
NEGative| POSitive

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

6.3 Diagnostic

class DiagnosticCls

Diagnostic 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.clone()
```

Subgroups

6.3.1 Point

SCPI Command :

```
DIAGnostic:POINt:CATalog
```

class PointCls

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

get_catalog() → List[str]

```
# SCPI: DIAGnostic:POINt:CATalog
value: List[str] = driver.diagnostic.point.get_catalog()
```

Queries the test points available in the instrument.

return
diag_poin_id_cat: No help available

6.4 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 RsSgt.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

6.4.1 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

6.4.1.1 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 RsSgt.MassMemory.currentDirectory command.

return
file_count: integer Number of files.

6.4.2 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

6.4.2.1 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 RsSgt.MassMemory.currentDirectory command.

return
directory_count: integer Number of parent and subdirectories.

6.4.3 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

6.4.3.1 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

6.4.4 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

6.4.4.1 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

6.5 Output

SCPI Command :

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

class OutputCls

Output commands group definition. 6 total commands, 3 Subgroups, 1 group commands

get_amode() → AttMode

```
# SCPI: OUTPut<HW>:AMODe
value: enums.AttMode = driver.output.get_amode()
```

Switches the mode of the attenuator at the RF output.

return
amode: AUTO| FIXed| APASsive AUTO The attenuator is switched automatically. The level settings are made in the full range. APASsive The attenuator is switched automatically. The level settings are made only for the passive reference circuits. The high-level ranges are not available. FIXed The level settings are made without switching the attenuator. When this operating mode is switched on, the attenuator is fixed to its current position and the resulting variation range is defined.

set_amode(*amode: AttMode*) → None

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

Switches the mode of the attenuator at the RF output.

param amode
AUTO| FIXed| APASsive AUTO The attenuator is switched automatically. The level settings are made in the full range. APASsive The attenuator is switched automatically. The level settings are made only for the passive reference circuits. The high-level ranges are not available. FIXed The level settings are made without switching the attenuator. When this operating mode is switched on, the attenuator is fixed to its current position and the resulting variation range is defined.

Cloning the Group

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

Subgroups

6.5.1 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

6.5.1.1 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 minimum level which can be set without the attenuator being adjusted (Attenuator FIXed) .

return
lower: float

get_upper() → float

```
# SCPI: OUTPut<HW>:AFIXed:RANGe:UPPer
value: float = driver.output.afixed.range.get_upper()
```

Queries the maximum level which can be set without the attenuator being adjusted (Attenuator FIXed) .

return
upper: float

6.5.2 Protection

SCPI Command :

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

class ProtectionCls

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

clear() → None

```
# SCPI: OUTPut<HW>:PROTection:CLEar
driver.output.protection.clear()
```

No command help available

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

```
# SCPI: OUTPut<HW>:PROTection:CLEar
driver.output.protection.clear_with_opc()
```

No command help available

Same as clear, but waits for the operation to complete before continuing further. Use the RsSgt.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 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() → PowOutpPonMode

```
# SCPI: OUTPut<HW>:[STATe]:PON
value: enums.PowOutpPonMode = driver.output.state.get_pon()
```

Selects the state of the RF output when the instrument is switched on.

return
pon: OFF| UNCHanged OFF Deactivates the output when the instrument is switched on ([RF off]) . UNCHanged Restores the initial state of the RF output before the last turn off.

get_value() → bool

```
# SCPI: OUTPut<HW>:[STATe]
value: bool = driver.output.state.get_value()
```

Activates the RF output signal ([RF on/off]) .

return
state: 1| ON| 0| OFF

set_pon(*pon: PowOutpPonMode*) → None

```
# SCPI: OUTPut<HW>:[STATe]:PON
driver.output.state.set_pon(pon = enums.PowOutpPonMode.OFF)
```

Selects the state of the RF output when the instrument is switched on.

param pon
OFF| UNCHanged OFF Deactivates the output when the instrument is switched on ([RF off]) . UNCHanged Restores the initial state of the RF output before the last turn off.

set_value(*state: bool*) → None

```
# SCPI: OUTPut<HW>:[STATe]
driver.output.state.set_value(state = False)
```

Activates the RF output signal ([RF on/off]) .

param state
1| ON| 0| OFF

6.6 Sconfiguration

SCPI Command :

```
SCONfiguration:MODE
```

class SconfigurationCls

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

get_mode() → BbSystemConfiguration

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

Switches between standard mode and ARB mode for envelope tracking.

return
configuration: STANdard| AFETracking STANdard Standard mode used for signal generation. AFETracking ARB foe Envelope Tracking: enables the usage of an extra baseband for enabling the envelope tracking ARB generation.

set_mode(*configuration: BbSystemConfiguration*) → None

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

Switches between standard mode and ARB mode for envelope tracking.

param configuration
STANdard| AFETracking STANdard Standard mode used for signal generation. AFETracking ARB foe Envelope Tracking: enables the usage of an extra baseband for enabling the envelope tracking ARB generation.

6.7 Source

SCPI Command :

```
[SOURce<HW>]:OPMode
```

class SourceCls

Source commands group definition. 64 total commands, 9 Subgroups, 1 group commands

get_op_mode() → OpMode

```
# SCPI: [SOURce<HW>]:OPMode
value: enums.OpMode = driver.source.get_op_mode()
```

Sets the operation mode.

return
op_mode: NORMal| BBBYpass NORMal normal operation BBBYpass Baseband bypass mode

set_op_mode(*op_mode: OpMode*) → None

```
# SCPI: [SOURce<HW>]:OPMode
driver.source.set_op_mode(op_mode = enums.OpMode.BBBYpass)
```

Sets the operation mode.

param op_mode
NORMal| BBBYpass NORMal normal operation BBBYpass Baseband bypass mode

Cloning the Group

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

Subgroups

6.7.1 Bb

SCPI Commands :

```
[SOURce<HW>]:BB:FOFFset
[SOURce<HW>]:BB:POFFset
```

class BbCls

Bb commands group definition. 8 total commands, 1 Subgroups, 2 group commands

get_foffset() → float

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

Sets the frequency offset for the baseband signal. The offset affects the signal on the baseband block output. It shifts the useful baseband signal in the center frequency.

return
foffset: float

get_poffset() → float

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

Sets the relative phase offset of the baseband signal. The phase offset affects the signal of the 'Baseband Block' output.

return
ph_offset: float

set_foffset(*foffset: float*) → None

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

Sets the frequency offset for the baseband signal. The offset affects the signal on the baseband block output. It shifts the useful baseband signal in the center frequency.

param foffset
float

set_poffset(*ph_offset: float*) → None

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

Sets the relative phase offset of the baseband signal. The phase offset affects the signal of the 'Baseband Block' output.

param ph_offset
float

Cloning the Group

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

Subgroups

6.7.1.1 Impairment

class ImpairmentCls

Impairment commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.1.1.1 Bbmm<IqConnector>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.source.bb.impairment.bbmm.repcap_iqConnector_get()
driver.source.bb.impairment.bbmm.repcap_iqConnector_set(repcap.IqConnector.Nr1)
```

class BbmmCls

Bbmm commands group definition. 1 total commands, 1 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.bbmm.clone()
```

Subgroups

6.7.1.1.1.1 Poffset

SCPI Command :

```
[SOURce]:BB:IMPairment:BBMM<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:BBMM<CH>:POFFset
value: float = driver.source.bb.impairment.bbmm.poffset.get(iqConnector =␣
↪repcap.IqConnector.Default)
```

No command help available

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

return
phase_offset: No help available

set(*phase_offset: float*, *iqConnector=IqConnector.Default*) → None

```
# SCPI: [SOURce]:BB:IMPairment:BBMM<CH>:POFFset
driver.source.bb.impairment.bbmm.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 'Bbmm')

6.7.1.1.2 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. 5 total commands, 4 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

6.7.1.1.2.1 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

6.7.1.1.2.2 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)
```

Sets the ratio of I modulation to Q modulation (amplification imbalance) of the corresponding digital I/Q channel. The input may be either in dB or %. The resolution is 0.001 dB, an input in percent is rounded to the closest valid value in dB. A query returns the value in dB.

INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

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)
```

Sets the ratio of I modulation to Q modulation (amplification imbalance) of the corresponding digital I/Q channel. The input may be either in dB or %. The resolution is 0.001 dB, an input in percent is rounded to the closest valid value in dB. A query returns the value in dB.

INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

param ipartq_ratio
float Range: -1 to 1

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

6.7.1.1.2.3 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

6.7.1.1.2.4 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)
```

Determines the leakage amplitude of the I or Q-signal component of the corresponding stream.
INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

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)
```

Determines the leakage amplitude of the I or Q-signal component of the corresponding stream.
INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

param ipart
float Range: -10 to 10

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

6.7.1.1.2.5 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)
```

Determines the leakage amplitude of the I or Q-signal component of the corresponding stream.
INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

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)
```

Determines the leakage amplitude of the I or Q-signal component of the corresponding stream.
INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

param qpart
float Range: -10 to 10

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

6.7.1.1.2.6 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

6.7.1.1.2.7 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)
```

Sets the quadrature offset. A positive quadrature offset results in a phase angle greater than 90 degrees.
INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

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

return
angle: float Range: -10 to 10

`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)
```

Sets the quadrature offset. A positive quadrature offset results in a phase angle greater than 90 degrees.
INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

param angle
float Range: -10 to 10

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

6.7.1.1.2.8 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)
```

Activates the impairment or correction values LEAKage, QUADrature and IQRatio for the corresponding stream.
INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

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

return
state: 0| 1| OFF| ON

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)
```

Activates the impairment or correction values LEAKage, QUADrature and IQRatio for the corresponding stream.

INTRO_CMD_HELP: The suffix <ch> has the following values:

- <ch>= 0: I/Q Analog Outputs
- <ch>= 1: I/Q Modulator Digital Impairments

param state
0| 1| OFF| ON

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

6.7.2 Bbin

SCPI Command :

```
[SOURce<HW>]:BBIN:ODELay
```

class BbinCls

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

get_odelay() → float

```
# SCPI: [SOURce<HW>]:BBIN:ODELay
value: float = driver.source.bbin.get_odelay()
```

Seds the output delay of the external baseband signal.

return
delay: float Range: 0 to 1

set_odelay(*delay: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:ODELay
driver.source.bbin.set_odelay(delay = 1.0)
```

Seds the output delay of the external baseband signal.

param delay
float Range: 0 to 1

Cloning the Group

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

Subgroups

6.7.2.1 SymbolRate

SCPI Command :

```
[SOURce<HW>]:BBIN:SRATe:[ACTual]
```

class SymbolRateCls

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

get_actual() → float

```
# SCPI: [SOURce<HW>]:BBIN:SRATe:[ACTual]
value: float = driver.source.bbin.symbolRate.get_actual()
```

Sets the sample rate of the external digital baseband signal.

return
actual: float Range: 25E6 to 250E6

set_actual(*actual: float*) → None

```
# SCPI: [SOURce<HW>]:BBIN:SRATe:[ACTual]
driver.source.bbin.symbolRate.set_actual(actual = 1.0)
```

Sets the sample rate of the external digital baseband signal.

param actual
float Range: 25E6 to 250E6

6.7.3 Frequency

SCPI Command :

```
[SOURce]:FREQuency:OFFSet
```

class FrequencyCls

Frequency commands group definition. 9 total commands, 2 Subgroups, 1 group commands

get_offset() → float

```
# SCPI: [SOURce]:FREQuency:OFFSet
value: float = driver.source.frequency.get_offset()
```

Sets a frequency offset, for example include the frequency shift of downstream instrument. Note: Enabled frequency offset affects the result of the query SOURce:FREQuency:CW? The query returns the frequency, including frequency offset.

return
offset: float Range: -3e9 to 3e9

set_offset(*offset: float*) → None

```
# SCPI: [SOURce]:FREQuency:OFFSet
driver.source.frequency.set_offset(offset = 1.0)
```

Sets a frequency offset, for example include the frequency shift of downstream instrument. Note: Enabled frequency offset affects the result of the query SOURce:FREQuency:CW? The query returns the frequency, including frequency offset.

param offset
float Range: -3e9 to 3e9

Cloning the Group

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

Subgroups

6.7.3.1 Cw

SCPI Commands :

```
[SOURce<HW>]:FREQuency:[CW]:RCL
[SOURce<HW>]:FREQuency:[CW]
```

class CwCls

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

get_recall() → InclExcl

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:RCL
value: enums.InclExcl = driver.source.frequency.cw.get_recall()
```

No command help available

return
rcl_excl_freq: No help available

get_value() → float

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]
value: float = driver.source.frequency.cw.get_value()
```

Sets the RF frequency at the RF output connector of the selected instrument. Note: Enabled frequency offset affects the result of this query. The query returns the frequency, including frequency offset.

return
cw: float

set_recall(*rcl_excl_freq: InclExcl*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:RCL
driver.source.frequency.cw.set_recall(rcl_excl_freq = enums.InclExcl.EXCLude)
```

No command help available

param rcl_excl_freq
No help available

set_value(*cw: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]
driver.source.frequency.cw.set_value(cw = 1.0)
```

Sets the RF frequency at the RF output connector of the selected instrument. Note: Enabled frequency offset affects the result of this query. The query returns the frequency, including frequency offset.

param cw
float

Cloning the Group

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

Subgroups

6.7.3.1.1 Step

SCPI Commands :

```
[SOURce<HW>]:FREQuency:[CW]:STEP:MODE
[SOURce<HW>]:FREQuency:[CW]:STEP:[INCRement]
```

class StepCls

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

get_increment() → float

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:STEP:[INCRement]
value: float = driver.source.frequency.cw.step.get_increment()
```

No command help available

return
freq_step: No help available

get_mode() → FreqStepMode

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:STEP:MODE
value: enums.FreqStepMode = driver.source.frequency.cw.step.get_mode()
```

No command help available

return
freq_step_mode: No help available

set_increment(*freq_step: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:STEP:[INCRement]
driver.source.frequency.cw.step.set_increment(freq_step = 1.0)
```

No command help available

param freq_step
No help available

set_mode(*freq_step_mode: FreqStepMode*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[CW]:STEP:MODE
driver.source.frequency.cw.step.set_mode(freq_step_mode = enums.FreqStepMode.
↪DECimal)
```

No command help available

param freq_step_mode
No help available

6.7.3.2 Fixed

SCPI Commands :

```
[SOURce<HW>]:FREQuency:[FIXed]:RCL
[SOURce<HW>]:FREQuency:[FIXed]
```

class FixedCls

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

get_recall() → InclExcl

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:RCL
value: enums.InclExcl = driver.source.frequency.fixed.get_recall()
```

No command help available

return
rcl_excl_freq: No help available

get_value() → float

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]
value: float = driver.source.frequency.fixed.get_value()
```

Sets the RF frequency at the RF output connector of the selected instrument. Note: Enabled frequency offset affects the result of this query. The query returns the frequency, including frequency offset.

return
cw: float

set_recall(*rcl_excl_freq: InclExcl*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:RCL
driver.source.frequency.fixed.set_recall(rcl_excl_freq = enums.InclExcl.EXCLude)
```

No command help available

param rcl_excl_freq
No help available

set_value(*cw: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]
driver.source.frequency.fixed.set_value(cw = 1.0)
```

Sets the RF frequency at the RF output connector of the selected instrument. Note: Enabled frequency offset affects the result of this query. The query returns the frequency, including frequency offset.

param cw
float

Cloning the Group

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

Subgroups

6.7.3.2.1 Step

SCPI Commands :

```
[SOURce<HW>]:FREQuency:[FIXed]:STEP:MODE
[SOURce<HW>]:FREQuency:[FIXed]:STEP:[INCRement]
```

class StepCls

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

get_increment() → float

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:STEP:[INCRement]
value: float = driver.source.frequency.fixed.step.get_increment()
```

No command help available

return
freq_step: No help available

get_mode() → FreqStepMode

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:STEP:MODE
value: enums.FreqStepMode = driver.source.frequency.fixed.step.get_mode()
```

No command help available

return
freq_step_mode: No help available

set_increment(*freq_step: float*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:STEP:[INCRement]
driver.source.frequency.fixed.step.set_increment(freq_step = 1.0)
```

No command help available

param freq_step
No help available

set_mode(*freq_step_mode: FreqStepMode*) → None

```
# SCPI: [SOURce<HW>]:FREQuency:[FIXed]:STEP:MODE
driver.source.frequency.fixed.step.set_mode(freq_step_mode = enums.FreqStepMode.
↪DECimal)
```

No command help available

param freq_step_mode
No help available

6.7.4 InputPy

class InputPyCls

InputPy commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.4.1 Modext

SCPI Command :

```
[SOURce]:INPut:MODext:IMPedance
```

class ModextCls

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

get_impedance() → ImpG50G10K

```
# SCPI: [SOURce]:INPut:MODext:IMPedance
value: enums.ImpG50G10K = driver.source.inputPy.modext.get_impedance()
```

No command help available

return
impedance: No help available

set_impedance(*impedance: ImpG50G10K*) → None

```
# SCPI: [SOURce]:INPut:MODext:IMPedance
driver.source.inputPy.modext.set_impedance(impedance = enums.ImpG50G10K.G10K)
```

No command help available

param impedance
No help available

6.7.4.2 Trigger

SCPI Commands :

```
[SOURce<HW>]:INPut:TRIGger:IMPedance
[SOURce<HW>]:INPut:TRIGger:LEVel
```

class TriggerCls

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

get_impedance() → InputImpRf

```
# SCPI: [SOURce<HW>]:INPut:TRIGger:IMPedance
value: enums.InputImpRf = driver.source.inputPy.trigger.get_impedance()
```

No command help available

return
impedance: No help available

get_level() → float

```
# SCPI: [SOURce<HW>]:INPut:TRIGger:LEVel
value: float = driver.source.inputPy.trigger.get_level()
```

No command help available

return
level: No help available

set_impedance(*impedance: InputImpRf*) → None

```
# SCPI: [SOURce<HW>]:INPut:TRIGger:IMPedance
driver.source.inputPy.trigger.set_impedance(impedance = enums.InputImpRf.G10K)
```

No command help available

param impedance
No help available

Cloning the Group

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

Subgroups

6.7.4.2.1 Bband

SCPI Command :

```
[SOURce<HW>]:INPut:TRIGger:[BBANd]:SLOPe
```

class BbandCls

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

get_slope() → SlopeType

```
# SCPI: [SOURce<HW>]:INPut:TRIGger:[BBANd]:SLOPe
value: enums.SlopeType = driver.source.inputPy.trigger.bband.get_slope()
```

No command help available

return
slope: No help available

set_slope(*slope: SlopeType*) → None

```
# SCPI: [SOURce<HW>]:INPut:TRIGger:[BBANd]:SLOPe
driver.source.inputPy.trigger.bband.set_slope(slope = enums.SlopeType.NEGative)
```

No command help available

param slope
No help available

6.7.5 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. 10 total commands, 1 Subgroups, 4 group commands

get_crest_factor() → float

```
# SCPI: [SOURce<HW>]:IQ:CREStfactor
value: float = driver.source.iq.get_crest_factor()
```

Sets the crest factor of the IQ modulation signal.

return
crest_factor: float Range: 0 to 80

get_source() → IqMode

```
# SCPI: [SOURce<HW>]:IQ:SOURce
value: enums.IqMode = driver.source.iq.get_source()
```

Sets the input signal for the I/Q modulator.

return
source: ANALog| BASeband

get_state() → bool

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

Switches the I/Q modulation on and off.

return
state: 1| ON| 0| OFF

get_wb_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:WBSTate
value: bool = driver.source.iq.get_wb_state()
```

Selects optimized settings for wideband modulation signals.

return
state: 1| ON| 0| OFF

set_crest_factor(*crest_factor: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:CRESt:factor
driver.source.iq.set_crest_factor(crest_factor = 1.0)
```

Sets the crest factor of the IQ modulation signal.

param crest_factor
float Range: 0 to 80

set_source(*source: IqMode*) → None

```
# SCPI: [SOURce<HW>]:IQ:SOURce
driver.source.iq.set_source(source = enums.IqMode.ANALog)
```

Sets the input signal for the I/Q modulator.

param source
ANALog| BASeband

set_state(*state: bool*) → None

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

Switches the I/Q modulation on and off.

param state
1| ON| 0| OFF

set_wb_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:WBSTate
driver.source.iq.set_wb_state(state = False)
```

Selects optimized settings for wideband modulation signals.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.7.5.1 Impairment

SCPI Command :

```
[SOURce<HW>]:IQ:IMPairment:[STATe]
```

class ImpairmentCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:[STATe]
value: bool = driver.source.iq.impairment.get_state()
```

Activates/ deactivates the three impairment or correction values LEAKage, QUADrature and IQRatio for the baseband signal prior to input into the I/Q modulator.

return
state: 1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:[STATe]
driver.source.iq.impairment.set_state(state = False)
```

Activates/ deactivates the three impairment or correction values LEAKage, QUADrature and IQRatio for the baseband signal prior to input into the I/Q modulator.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.7.5.1.1 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()
```

Sets the ratio of I modulation to Q modulation (amplification 'imbalance') . The input may be either in dB or %. The resolution is 0.001 dB, an input in percent is rounded to the closest valid value in dB. A query returns the value in dB.

return
ipartq_ratio: No help available

set_magnitude(*ipartq_ratio: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:IQRatio:[MAGNitude]
driver.source.iq.impairment.iqRatio.set_magnitude(ipartq_ratio = 1.0)
```

Sets the ratio of I modulation to Q modulation (amplification 'imbalance') . The input may be either in dB or %. The resolution is 0.001 dB, an input in percent is rounded to the closest valid value in dB. A query returns the value in dB.

param ipartq_ratio
float Range: -1 to 1

6.7.5.1.2 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()
```

Sets the carrier leakage amplitude for the I-signal component.

return
ipart: No help available

get_qcomponent() → float

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:LEAKage:Q
value: float = driver.source.iq.impairment.leakage.get_qcomponent()
```

Sets the carrier leakage amplitude for the Q-signal component.

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)
```

Sets the carrier leakage amplitude for the I-signal component.

param ipart
float Range: -10 to 10

set_qcomponent(*qpart: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:LEAKage:Q
driver.source.iq.impairment.leakage.set_qcomponent(qpart = 1.0)
```

Sets the carrier leakage amplitude for the Q-signal component.

param qpart
float Range: -10 to 10

6.7.5.1.3 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()
```

Sets the quadrature offset for the digital I/Q signal.

return
angle: float Range: -10 to 10

set_angle(*angle: float*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:QUADrature:[ANGLe]
driver.source.iq.impairment.quadrature.set_angle(angle = 1.0)
```

Sets the quadrature offset for the digital I/Q signal.

param angle
float Range: -10 to 10

6.7.5.1.4 Swap

SCPI Command :

```
[SOURce<HW>]:IQ:IMPairment:SWAP:[STATe]
```

class SwapCls

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

get_state() → bool

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:SWAP:[STATe]
value: bool = driver.source.iq.impairment.swap.get_state()
```

When set to ON, this command swaps the I and Q channel for an external modulation signal.

return
state: 1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:IQ:IMPairment:SWAP:[STATe]
driver.source.iq.impairment.swap.set_state(state = False)
```

When set to ON, this command swaps the I and Q channel for an external modulation signal.

param state
1| ON| 0| OFF

6.7.6 Loscillator

SCPI Command :

```
[SOURce<HW>]:LOSCillator:SOURce
```

class LoscillatorCls

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

get_source() → SourceInt

```
# SCPI: [SOURce<HW>]:LOSCillator:SOURce
value: enums.SourceInt = driver.source.loscillator.get_source()
```

Selects the source of the local oscillator signal.

return
source: INTernal| EXTernal INT: use built in oscillator; EXT: use signal at [LO/ REF IN] connector

set_source(*source: SourceInt*) → None

```
# SCPI: [SOURce<HW>]:LOSCillator:SOURce
driver.source.loscillator.set_source(source = enums.SourceInt.EXTernal)
```

Selects the source of the local oscillator signal.

param source
INTernal| EXTernal INT: use built in oscillator; EXT: use signal at [LO/ REF IN] connector

6.7.7 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()
```

Specifies the phase variation relative to the current phase.

return
phase: float Range: -360 to 360, Unit: DEG

set_value(*phase: float*) → None

```
# SCPI: [SOURce<HW>]:PHASe
driver.source.phase.set_value(phase = 1.0)
```

Specifies the phase variation relative to the current phase.

param phase
float Range: -360 to 360, Unit: DEG

Cloning the Group

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

Subgroups

6.7.7.1 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()
```

Adopts the phase set with command [:SOURce]:PHASe as the current phase.

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

```
# SCPI: [SOURce<HW>]:PHASe:REFerence
driver.source.phase.reference.set_with_opc()
```

Adopts the phase set with command [:SOURce]:PHASe as the current phase.

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

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

6.7.8 Power

SCPI Commands :

```
[SOURce<HW>]:POWer:LMODe
[SOURce<HW>]:POWer:PEP
[SOURce<HW>]:POWer:POWer
[SOURce<HW>]:POWer:SCHaracteristic
```

class PowerCls

Power commands group definition. 24 total commands, 6 Subgroups, 4 group commands

get_lmode() → PowLevMode

```
# SCPI: [SOURce<HW>]:POWer:LMODe
value: enums.PowLevMode = driver.source.power.get_lmode()
```

Selects the level mode.

return
lev_mode: NORMal| LOWNoise| LOWDistortion NORM automatic selection of the best settings LNOISe settings for lowest noise LDIStortion settings for lowest distortions

get_pep() → float

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

Queries the RF signal peak envelope power.

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]:POWer[:LEVel][:IMMediate][:AMPLitude] sets the level of the 'Level' display, that means the level containing offset.

return
amplitude: float Range: -20 to 25

get_scharacteristic() → PowLevBehaviour

```
# SCPI: [SOURce<HW>]:POWer:SCHaracteristic
value: enums.PowLevBehaviour = driver.source.power.get_scharacteristic()
```

Selects the characteristic for the level setting.

return
characteristic: AUTO| UNINterrupted| CVSWr| USER| MONotone UNINterrupted uninterrupted level setting CVSWr constant-VSWR MONotone strictly monotone

set_lmode(*lev_mode: PowLevMode*) → None

```
# SCPI: [SOURce<HW>]:POWer:LMODe
driver.source.power.set_lmode(lev_mode = enums.PowLevMode.LOWDistortion)
```

Selects the level mode.

param lev_mode
NORMal| LOWNoise| LOWDistortion NORM automatic selection of the best settings LNOISe settings for lowest noise LDIStortion settings for lowest distortions

set_power(*amplitude: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:POWer
driver.source.power.set_power(amplitude = 1.0)
```

Sets the level at the RF output connector. This value does not consider a specified offset. The command [:SOURce]:POWer[:LEVel][:IMMediate][:AMPLitude] sets the level of the 'Level' display, that means the level containing offset.

param amplitude
float Range: -20 to 25

set_scharacteristic(*characteristic: PowLevBehaviour*) → None

```
# SCPI: [SOURce<HW>]:POWer:SCHaracteristic
driver.source.power.set_scharacteristic(characteristic = enums.PowLevBehaviour.
↪AUTO)
```

Selects the characteristic for the level setting.

param characteristic
AUTO| UNINterrupted| CVSWr| USER| MONotone UNINterrupted uninterrupted level setting CVSWr constant-VSWR MONotone strictly monotone

Cloning the Group

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

Subgroups

6.7.8.1 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() → CalPowDetAtt

```
# SCPI: [SOURce<HW>]:POWer:ALC:DSENsitivity
value: enums.CalPowDetAtt = driver.source.power.alc.get_dsensitivity()
```

Sets the power detector sensitivity. Used for compatibility reasons only.

return
sensitivity: OFF| LOW| MED| HIGH

get_state() → PowAlcState

```
# SCPI: [SOURce<HW>]:POWer:ALC:[STATe]
value: enums.PowAlcState = driver.source.power.alc.get_state()
```

Activates/deactivates automatic level control.

return
state: 1| OFFTable| OFF| ONTable| AUTO| ON

set_dsensitivity(*sensitivity: CalPowDetAtt*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:DSENsitivity
driver.source.power.alc.set_dsensitivity(sensitivity = enums.CalPowDetAtt.HIGH)
```

Sets the power detector sensitivity. Used for compatibility reasons only.

param sensitivity
OFF| LOW| MED| HIGH

set_state(*state: PowAlcState*) → None

```
# SCPI: [SOURce<HW>]:POWer:ALC:[STATe]
driver.source.power.alc.set_state(state = enums.PowAlcState._1)
```

Activates/deactivates automatic level control.

param state
1| OFFTable| OFF| ONTable| AUTO| ON

Cloning the Group

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

Subgroups

6.7.8.1.1 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()
```

Briefly activates level control for correction purposes.

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

```
# SCPI: [SOURce<HW>]:POWer:ALC:SONCe
driver.source.power.alc.sonce.set_with_opc()
```

Briefly activates level control for correction purposes.

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

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

6.7.8.2 Attenuation

SCPI Command :

```
[SOURce<HW>]:POWer:ATTenuation:DIGital
```

class AttenuationCls

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

get_digital() → float

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:DIGital
value: float = driver.source.power.attenuation.get_digital()
```

Sets a relative attentuation value for the baseband signal.

return
att_digital: float Range: 0 to 80, Unit: dB

set_digital(*att_digital: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:DIGital
driver.source.power.attenuation.set_digital(att_digital = 1.0)
```

Sets a relative attentuation value for the baseband signal.

param att_digital
float Range: 0 to 80, Unit: dB

Cloning the Group

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

Subgroups

6.7.8.2.1 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() → PowRfOffMode

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:RFOFf:MODE
value: enums.PowRfOffMode = driver.source.power.attenuation.rfOff.get_mode()
```

Determines the attenuator's state after the instrument is switched on.

return
mode: MAX| FATTenuated| FIXed| UNCHanged MAX = FATTenuated Sets attenuation to maximum when the RF signal is switched off. This setting is recommended for applications that require a high level of noise suppression. FIXed = UNCHanged Retains the current setting and keeps the output impedance constant during RF off.

set_mode(*mode: PowRfOffMode*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:RFOFf:MODE
driver.source.power.attenuation.rfOff.set_mode(mode = enums.PowRfOffMode.FIXed)
```

Determines the attenuator's state after the instrument is switched on.

param mode
MAX| FATTenuated| FIXed| UNCHanged MAX = FATTenuated Sets attenuation to maximum when the RF signal is switched off. This setting is recommended for applications that require a high level of noise suppression. FIXed = UNCHanged Retains the current setting and keeps the output impedance constant during RF off.

6.7.8.2.2 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()
```

Sets the switch-over offset value of the attenuator.

return
offset: float Range: -10 to 10

set_offset(*offset: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:ATTenuation:SOVer:[OFFSet]
driver.source.power.attenuation.sover.set_offset(offset = 1.0)
```

Sets the switch-over offset value of the attenuator.

param offset
float Range: -10 to 10

6.7.8.3 Level

class LevelCls

Level 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.power.level.clone()
```

Subgroups

6.7.8.3.1 Immediate

SCPI Commands :

```
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:OFFSet
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:[AMPLitude]
```

class ImmediateCls

Immediate commands group definition. 3 total commands, 0 Subgroups, 3 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 at the RF output connector of the instrument.

return
amplitude: float Range: -120 to 25, Unit: dBm

get_offset() → float

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:OFFSet
value: float = driver.source.power.level.immediate.get_offset()
```

Specifies the constant level offset of a downstream attenuator/amplifier. If a level offset is entered, the level entered with POWer no longer corresponds to the RF output level. The following correlation applies: POWer = RF output level + POWer:OFFSet. Entering a level offset does not change the RF output level, but rather the query value of POWer. Only dB is permitted as the unit here. The linear units (V, W, etc.) are not permitted.

return
offset: float Range: -100 to 100

get_recall() → InclExcl

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
value: enums.InclExcl = driver.source.power.level.immediate.get_recall()
```

No command help available

return
rcl_excl_pow: No help available

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 at the RF output connector of the instrument.

param amplitude
float Range: -120 to 25, Unit: dBm

set_offset(*offset: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:OFFSet
driver.source.power.level.immediate.set_offset(offset = 1.0)
```

Specifies the constant level offset of a downstream attenuator/amplifier. If a level offset is entered, the level entered with POWer no longer corresponds to the RF output level. The following correlation applies: POWer = RF output level + POWer:OFFSet. Entering a level offset does not change the RF output level, but rather the query value of POWer. Only dB is permitted as the unit here. The linear units (V, W, etc.) are not permitted.

param offset
float Range: -100 to 100

set_recall(*rcl_excl_pow: InclExcl*) → None

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
driver.source.power.level.immediate.set_recall(rcl_excl_pow = enums.InclExcl.
↪EXCLude)
```

No command help available

param rcl_excl_pow
No help available

6.7.8.4 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()
```

Sets the upper limit of the RF signal power. An instrument preset does not affect this value. A factory preset (method RsSgt.System.Fpreset.set) sets the specified factory upper limit value.

return
amplitude: float Range: -120 to 25

set_amplitude(*amplitude: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:LIMit:[AMPLitude]
driver.source.power.limit.set_amplitude(amplitude = 1.0)
```

Sets the upper limit of the RF signal power. An instrument preset does not affect this value. A factory preset (method RsSgt.System.Fpreset.set) sets the specified factory upper limit value.

param amplitude
float Range: -120 to 25

6.7.8.5 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 minimum/maximum level range in the current level mode

return
lower: float

get_upper() → float

```
# SCPI: [SOURce<HW>]:POWer:RANGe:UPPer
value: float = driver.source.power.range.get_upper()
```

Queries the minimum/maximum level range in the current level mode

return
upper: float

6.7.8.6 Servoing

SCPI Commands :

```
[SOURce<HW>]:POWer:SERVoing:SET
[SOURce<HW>]:POWer:SERVoing:STATe
[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. 8 total commands, 1 Subgroups, 6 group commands

class SetStruct

Structure for reading output parameters. Fields:

- Target: float: float Starts a single power servoing setting cycle, see 'About power servoing'. Sets the desired output power level required at the DUT, see[:SOURce]:POWer:SERVoing:TARGet.
- Start: enums.Test: 0| 1| RUNning| STOPped Queries the current status of the powerservoing procedure. 0|RUNning Procedure is ok and or running without errors. 1|STOPped Procedure is stopped due to an error.

get_set() → SetStruct

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:SET
value: SetStruct = driver.source.power.servoing.get_set()
```

Sets the target output power level and queries power servoing procedure status.

return
structure: for return value, see the help for SetStruct structure arguments.

get_state() → bool

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:STATe
value: bool = driver.source.power.servoing.get_state()
```

Activates/deactivates power servoing.

return
state: 1| ON| 0| OFF

get_target() → float

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TARGet
value: float = driver.source.power.servoing.get_target()
```

Sets the target output power level required at the DUT.

return
target_level: float Range: -120 to 25

get_test() → Test

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TEST
value: enums.Test = driver.source.power.servoing.get_test()
```

Queries the state of the power servoing procedure.

return
start: 0| 1| RUNning| STOPped

get_tolerance() → float

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TOLerance
value: float = driver.source.power.servoing.get_tolerance()
```

Sets the tolerance level interval, in which the 'Target' output power level of the DUT lies. A large tolerance accelerates the power servoing procedure but also reduces the accuracy of the target output power level.

return
tolerance: float Range: 0.01 to 3

get_tracking() → bool

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TRACking
value: bool = driver.source.power.servoing.get_tracking()
```

Activates/deactivates level tracking. Activation increases measurement accuracy but also measurement time.

return
state: 1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:STATe
driver.source.power.servoing.set_state(state = False)
```

Activates/deactivates power servoing.

param state
1| ON| 0| OFF

set_target(*target_level: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TARGet
driver.source.power.servoing.set_target(target_level = 1.0)
```

Sets the target output power level required at the DUT.

param target_level
float Range: -120 to 25

set_tolerance(*tolerance: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TOLerance
driver.source.power.servoing.set_tolerance(tolerance = 1.0)
```

Sets the tolerance level interval, in which the 'Target' output power level of the DUT lies. A large tolerance accelerates the power servoing procedure but also reduces the accuracy of the target output power level.

param tolerance
float Range: 0.01 to 3

set_tracking(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:TRACking
driver.source.power.servoing.set_tracking(state = False)
```

Activates/deactivates level tracking. Activation increases measurement accuracy but also measurement time.

param state
1| ON| 0| OFF

Cloning the Group

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

Subgroups

6.7.8.6.1 Sensor

SCPI Commands :

```
[SOURce<HW>]:POWer:SERVoing:SENSor:APERture
[SOURce<HW>]:POWer:SERVoing:SENSor
```

class SensorCls

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

get_aperture() → float

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:SENSor:APERture
value: float = driver.source.power.servoing.sensor.get_aperture()
```

Sets the aperture time (size of the acquisition interval) of the power sensor during power servoing.

return
aperture: float Range: 10e-6 to 100e-3

get_value() → PowSensWithUndef

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:SENSor
value: enums.PowSensWithUndef = driver.source.power.servoing.sensor.get_value()
```

Sets the power sensor as mapped with the remote command SLISt:ELEMent<ch>:MAPPing.

return
sensor: SENS1| SENS2| SENSor2| SENS3| SENSor3| SENS4| SENSor4| SENSor1| UNDefined

set_aperture(*aperture: float*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:SENSor:APERture
driver.source.power.servoing.sensor.set_aperture(aperture = 1.0)
```

Sets the aperture time (size of the acquisition interval) of the power sensor during power servoing.

param aperture
float Range: 10e-6 to 100e-3

set_value(*sensor: PowSensWithUndef*) → None

```
# SCPI: [SOURce<HW>]:POWer:SERVoing:SENSor
driver.source.power.servoing.sensor.set_value(sensor = enums.PowSensWithUndef.
↪SENS1)
```

Sets the power sensor as mapped with the remote command SLISt:ELEMent<ch>:MAPPing.

param sensor
SENS1| SENS2| SENSor2| SENS3| SENSor3| SENS4| SENSor4| SENSor1| UNDefined

6.7.9 Roscillator

SCPI Command :

```
[SOURce<HW>]:ROSCillator:SOURce
```

class RoscillatorCls

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

get_source() → SourceInt

```
# SCPI: [SOURce<HW>]:ROSCillator:SOURce
value: enums.SourceInt = driver.source.roscillator.get_source()
```

Select the reference oscillator signal source.

return
source: INTernal| EXTernal

set_source(*source: SourceInt*) → None

```
# SCPI: [SOURce<HW>]:ROSCillator:SOURce
driver.source.roscillator.set_source(source = enums.SourceInt.EXTernal)
```

Select the reference oscillator signal source.

param source
INTernal| EXTernal

Cloning the Group

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

Subgroups

6.7.9.1 External

SCPI Command :

```
[SOURce<HW>]:ROSCillator:EXTernal:FREQuency
```

class ExternalCls

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

get_frequency() → RoscFreqExt

```
# SCPI: [SOURce<HW>]:ROSCillator:EXTernal:FREQuency
value: enums.RoscFreqExt = driver.source.roscillator.external.get_frequency()
```

Selects the frequency of the external reference.

return
ext_freq: 10MHZ| 100MHZ| 1000MHZ| 13MHZ 13MHZ requires RF board with part number 1419.5308.02. To find out the RF board installed in the instrument: Select 'SGMA-GUI instrument name Setup Hardware Config' 'RF Assembly' Observe the part number of the assembly 'RfBoard'.

set_frequency(*ext_freq: RoscFreqExt*) → None

```
# SCPI: [SOURce<HW>]:ROSCillator:EXTernal:FREQuency
driver.source.roscillator.external.set_frequency(ext_freq = enums.RoscFreqExt._
↪1000MHZ)
```

Selects the frequency of the external reference.

param ext_freq
10MHZ| 100MHZ| 1000MHZ| 13MHZ 13MHZ requires RF board with part number 1419.5308.02. To find out the RF board installed in the instrument: Select 'SGMA-GUI instrument name Setup Hardware Config' 'RF Assembly' Observe the part number of the assembly 'RfBoard'.

6.7.9.2 Output

SCPI Command :

```
[SOURce<HW>]:ROSCillator:OUTPut:FREQuency
```

class OutputCls

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

get_frequency() → RoscFreqExt

```
# SCPI: [SOURce<HW>]:ROSCillator:OUTPut:FREQuency
value: enums.RoscFreqExt = driver.source.roscillator.output.get_frequency()
```

Selects the output for the reference oscillator signal.

return
output_freq: 10MHZ| 100MHZ| 1000MHZ| 13MHZ 13MHZ requires RF board with part number 1419.5308.02.

set_frequency(*output_freq: RoscFreqExt*) → None

```
# SCPI: [SOURce<HW>]:ROSCillator:OUTPut:FREQuency
driver.source.roscillator.output.set_frequency(output_freq = enums.RoscFreqExt._
↪1000MHZ)
```

Selects the output for the reference oscillator signal.

param output_freq
10MHZ| 100MHZ| 1000MHZ| 13MHZ 13MHZ requires RF board with part number 1419.5308.02.

6.8 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

6.8.1 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

6.8.1.1 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

6.8.1.1.1 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

6.8.1.1.2 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')

6.8.1.1.3 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

6.8.1.1.4 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')

6.8.1.1.5 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')

6.8.2 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 RsSgt.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 RsSgt.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

6.8.2.1 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

6.8.2.1.1 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

6.8.2.1.2 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')

6.8.2.1.3 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

6.8.2.1.4 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')

6.8.2.1.5 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')

6.8.3 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

6.9 System

SCPI Commands :

```
SYSTem:DID
SYSTem:PRESet
SYSTem:PRESet:ALL
SYSTem:PRESet:BASE
SYSTem:RESet
SYSTem:RESet:ALL
SYSTem:RESet:BASE
SYSTem:SREStore
SYSTem:SSAVe
SYSTem:TZONe
SYSTem:VERSion
```

class SystemCls

System commands group definition. 43 total commands, 8 Subgroups, 11 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_tzone() → str

```
# SCPI: SYSTem:TZONe
value: str = driver.system.get_tzone()
```

No command help available

return
pseudo_string: No help available

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 *RST command
    - The 'SGMA-GUI > Instrument Name > Preset' function. However, the command␣
↪does not close open GUI dialogs like the function does.

    :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

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

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

Cloning the Group

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

Subgroups

6.9.1 Date

SCPI Commands :

```
SYSTem:DATE:LOCal
SYSTem:DATE:UTC
```

class DateCls

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

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_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

6.9.2 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

6.9.3 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()
```

No command help available

return
device_footprint: No help available

set(*directory: str*) → None

```
# SCPI: SYSTem:DFPRint
driver.system.deviceFootprint.set(directory = 'abc')
```

No command help available

param directory
No help available

Cloning the Group

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

Subgroups

6.9.3.1 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

6.9.4 Error

SCPI Commands :

```
SYSTem:ERRor:ALL
SYSTem:ERRor:COUNt
```

class ErrorCls

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

get_all() → str

```
# SCPI: SYSTem:ERRor:ALL
value: str = driver.system.error.get_all()
```

Queries the error/event queue for all unread items and removes them from the queue.

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.

Cloning the Group

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

Subgroups

6.9.4.1 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 RsSgt.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 RsSgt.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.

6.9.5 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 RsSgt.utilities.opc_timeout_set() to set the timeout value.

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

6.9.6 Help

SCPI Command :

```
SYSTem:HELP:HEADers
```

class HelpCls

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

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

6.9.6.1 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

6.9.7 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

6.9.7.1 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

6.9.7.2 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()
```

No command help available

return
owner: No help available

6.9.7.3 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')
```

No command help available

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

6.9.7.4 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()
```

No command help available

return
success: No help available

Cloning the Group

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

Subgroups

6.9.7.4.1 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

6.9.7.5 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

6.9.8 Time

SCPI Commands :

```
SYSTem:TIME:LOCal
SYSTem:TIME:UTC
```

class TimeCls

Time commands group definition. 8 total commands, 2 Subgroups, 2 group commands

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_utc() → str

```
# SCPI: SYSTem:TIME:UTC
value: str = driver.system.time.get_utc()
```

No command help available

return
pseudo_string: No help available

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_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

6.9.8.1 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

6.9.8.1.1 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

6.9.8.2 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

6.9.8.2.1 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

6.10 Test

class TestCls

Test 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.test.clone()
```

Subgroups

6.10.1 All

SCPI Commands :

```
TEST:ALL:RESult
TEST:ALL:STARt
```

class AllCls

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

get_result() → Test

```
# SCPI: TEST:ALL:RESult
value: enums.Test = driver.test.all.get_result()
```

Queries the result of the performed selftest. Start the selftest with method RsSgt.Test.All.start.

return
result: 0| 1| RUNning| STOPped

start() → None

```
# SCPI: TEST:ALL:STARt
driver.test.all.start()
```

Starts the selftest. Use the command method RsSgt.Test.All.result to query the result.

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

```
# SCPI: TEST:ALL:STARt
driver.test.all.start_with_opc()
```

Starts the selftest. Use the command method RsSgt.Test.All.result to query the result.

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

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

6.10.2 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()
```

Enable/disable keyboard and LED test state.

return
state: 1| ON| 0| OFF

set_state(*state: bool*) → None

```
# SCPI: TEST:KEYBoard:[STATe]
driver.test.keyboard.set_state(state = False)
```

Enable/disable keyboard and LED test state.

param state
1| ON| 0| OFF

CHAPTER

SEVEN

RSSGT UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsSgt.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsSgt.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 RsSgt instance with new VISA session, the session gets a new thread lock. You can assign it to other RsSgt sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsSgt 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

RSSGT 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

RSSGT 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

T

U