RsNgx

Release 3.1.0.50

Rohde & Schwarz

Feb 03, 2023

CONTENTS:

CONTENTS:

CHAPTER

ONE

REVISION HISTORY

1.1 RsNgx

Rohde & Schwarz NGx Power Supply RsNgx instrument driver.

Basic Hello-World code:

```
from RsNgx import *

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

Check out the full documentation on ReadTheDocs.

Supported instruments: NGU, NGL, NGM, NGP

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

Documentation: https://RsNgx.readthedocs.io/

Examples: https://github.com/Rohde-Schwarz/Examples/tree/main/Misc/Python/RsNgx_ScpiPackage

1.1.1 Version history

Latest release notes summary: Docu Fixes

Version 3.1.0

- Docu Fixes

Version 3.1.0.49

- Rebuilt with new driver core

Version 3.1.0.48

- Update for NGP800 FW 2.015

Version 3.0.0.39

- First released version for NGU FW 3.0

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsNgx 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 RsNgx example:

```
"""RsNgx basic example - sets Voltage, Current limit and output state on two output␣
↪channels.
In comments above the calls, you see the SCPI commands sent. Notice, that the SCPI␣
↪commands
track the python interfaces."""

import time
from RsNgx import *

ngx = RsNgx('TCPIP::10.102.52.45::INSTR')
# Greetings, stranger...
print(f'Hello, I am: {ngx.utilities.idn_string}')
ngx.utilities.reset()

# Master switch for all the outputs - switch OFF
#   OUTPut:GENeral:STATe OFF
ngx.output.general.set_state(False)
```

(continues on next page)

(continued from previous page)

```
# Select and set Output 1
#   INSTrument:SELect 1
ngx.instrument.select.set(1)
#   SOURce:VOLTage:LEVel:IMMediate:AMPlitude 3.3
ngx.source.voltage.level.immediate.set_amplitude(3.3)
#   SOURce:CURRent:LEVel:IMMediate:AMPlitude 0.1
ngx.source.current.level.immediate.set_amplitude(0.1)
# Prepare for setting the output to ON with the master switch
#   OUTPut:SELect ON
ngx.output.set_select(True)

# Select and set Output 2
#   INSTrument:SELect 2
ngx.instrument.select.set(2)
#   SOURce:VOLTage:LEVel:IMMediate:AMPlitude 5.1
ngx.source.voltage.level.immediate.set_amplitude(5.1)
#   SOURce:CURRent:LEVel:IMMediate:AMPlitude 0.05
ngx.source.current.level.immediate.set_amplitude(0.05)
# Prepare for setting the output to ON with the master switch
#   OUTPut:SELect ON
ngx.output.set_select(True)

# The outputs are still OFF, they wait for this master switch:
#   OUTPut:GENeral:STATe ON
ngx.output.general.set_state(True)
# Insert a small pause to allow the instrument to settle the output
time.sleep(0.5)

#   INSTrument:SELect 1
ngx.instrument.select.set(1)
#                 READ?
measurement = ngx.read()
print(f'Measured values Output 1: {measurement.Voltage} V, {measurement.Current} A')

#   INSTrument:SELect 2
ngx.instrument.select.set(2)
#                 READ?
measurement = ngx.read()
print(f'Measured values Output 2: {measurement.Voltage} V, {measurement.Current} A')
ngx.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

RsNgx 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 RsNgx`

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 `RsNgx` 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 RsNgx offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsNgx needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsNgx package to your computer from the pypi.org: https://pypi.org/project/RsNgx/#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\RsNgx-3.1.0.50.tar`

2.3 Finding Available Instruments

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

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

from RsNgx import *

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

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

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

- Installed RsNgx Python module Version 3.1.0 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsNgx import *

# A good practice is to assure that you have a certain minimum version installed
RsNgx.assert_minimum_version('3.1.0')
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 = RsNgx(resource_string_1)

idn = driver.utilities.query_str('*IDN?')
print(f"\nHello, I am: '{idn}'")
print(f'RsNgx 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 2021.

Do not care about specialty of each session kind; RsNgx 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 = RsNgx('TCPIP::192.168.56.101::HISLIP', id_query=True, reset=True)
```

- Setting `id_query` to True (default is True) checks, whether your instrument can be used with the RsNgx 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 RsNgx allows you to choose which VISA to use:

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

from RsNgx import *

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

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

from RsNgx import *
```

(continues on next page)

(continued from previous page)

```
driver = RsNgx('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 = RsNgx('TCPIP::192.168.56.101::HISLIP', True, True, "Simulate=True")
```

More `option_string` tokens are separated by comma:

```
driver = RsNgx('TCPIP::192.168.56.101::HISLIP', 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 RsNgx objects:

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

from RsNgx import *

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

driver = RsNgx('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
"""
```

(continues on next page)

(continued from previous page)

```
from RsNgx import *

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

driver = RsNgx('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")
```

(continues on next page)

(continued from previous page)

```
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

RsNgx 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 RsNgx 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 RsNgx import *

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

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

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

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

RsNgx 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 RsNgx 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 = RsNgx('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
```

(continues on next page)

(continued from previous page)

```
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 RsNgx 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, RsNgx 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 RsNgx object
"""

import threading
from RsNgx import *


def execute(session):
```

(continues on next page)

(continued from previous page)

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


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

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsNgx import *


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

(continues on next page)

(continued from previous page)

```
    t = threading.Thread(target=execute, args=(driver1, 1, i,))
    t.start()
    threads.append(t)
    t = threading.Thread(target=execute, args=(driver2, 2, i,))
    t.start()
    threads.append(t)
print('All threads started')

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

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

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

Multiple instrument sessions accessed from multiple threads

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

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

Run the following example:

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

import threading
from RsNgx import *


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

(continues on next page)

(continued from previous page)

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

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

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

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

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

2.12 Logging

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

What can you actually do with the logger?

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

Let us take this basic example:

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

from RsNgx import *

driver = RsNgx('TCPIP::192.168.1.101::INSTR')
```

(continues on next page)

(continued from previous page)

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

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

Console output:

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

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

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

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

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

See the full logger help *here*.

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

```
driver = RsNgx('TCPIP::192.168.56.101::HISLIP', 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 RsNgx import *

driver = RsNgx('TCPIP::192.168.1.101::INSTR')
```

(continues on next page)

(continued from previous page)

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

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

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

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

driver.utilities.reset()

# Close the session
driver.close()

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

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

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

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

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

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```
from RsNgx import *

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

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

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

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

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

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

Console output:

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

Notice the following:

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

CHAPTER

THREE

ENUMS

3.1 ArbEndBehavior

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

3.2 ArbTrigMode

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

3.3 CalibrationType

```
# Example value:
value = enums.CalibrationType.CURR
# All values (4x):
CURR | IMP | NCUR | VOLT
```

3.4 DefaultStep

```
# Example value:
value = enums.DefaultStep.DEF
# All values (2x):
DEF | DEFault
```

3.5 DioFaultSource

```
# Example value:
value = enums.DioFaultSource.CC
# All values (6x):
CC | CR | CV | OUTPut | PROTection | SINK
```

3.6 DioOutSource

```
# Example value:
value = enums.DioOutSource.FORCed
# All values (3x):
FORCed | OUTPut | TRIGger
```

3.7 DioSignal

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

3.8 FastLogSampleRate

```
# Example value:
value = enums.FastLogSampleRate.S001k
# All values (6x):
S001k | S010k | S050k | S100 | S250k | S500k
```

3.9 FastLogTarget

```
# Example value:
value = enums.FastLogTarget.SCPI
# All values (2x):
SCPI | USB
```

3.10 Filename

```
# Example value:
value = enums.Filename.DEF
# All values (3x):
DEF | EXT | INT
```

3.11 HcpyFormat

```
# Example value:
value = enums.HcpyFormat.BMP
# All values (2x):
BMP | PNG
```

3.12 LogMode

```
# Example value:
value = enums.LogMode.COUNt
# All values (4x):
COUNt | DURation | SPAN | UNLimited
```

3.13 LowHigh

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

3.14 MinOrMax

```
# Example value:
value = enums.MinOrMax.MAX
# All values (4x):
MAX | MAXimum | MIN | MINimum
```

3.15 PrioMode

```
# Example value:
value = enums.PrioMode.CPM
# All values (2x):
CPM | VPM
```

3.16 TriggerChannel

```
# Example value:
value = enums.TriggerChannel.ALL
# All values (6x):
ALL | CH1 | CH2 | CH3 | CH4 | NONE
```

3.17 TriggerCondition

```
# First value:
value = enums.TriggerCondition.ANINput
# Last value:
value = enums.TriggerCondition.VMODe
# All values (19x):
ANINput | ARB | ARBGroup | ARBPoint | CMODe | ENABle | FUSE | ILEVel
INHibit | LOG | OPP | OTP | OUTPut | OVP | PLEVel | RAMP
STATistics | VLEVel | VMODe
```

3.18 TriggerDioSource

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

3.19 TriggerDirection

```
# Example value:
value = enums.TriggerDirection.INPut
# All values (2x):
INPut | OUTPut
```

3.20 TriggerOperMode

```
# Example value:
value = enums.TriggerOperMode.CC
# All values (5x):
CC | CR | CV | PROTection | SINK
```

3.21 TriggerSource

```
# Example value:
value = enums.TriggerSource.DIO
# All values (3x):
DIO | OMODe | OUTPut
```

3.22 UsbClass

```
# Example value:
value = enums.UsbClass.CDC
# All values (2x):
CDC | TMC
```

CHAPTER

FOUR

REPCAPS

4.1 Channel

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

4.2 Digitallo

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

CHAPTER

FIVE

EXAMPLES

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

```
"""RsNgx basic example - sets Voltage, Current limit and output state on two output␣
↪channels.
In comments above the calls, you see the SCPI commands sent. Notice, that the SCPI␣
↪commands
track the python interfaces."""

import time
from RsNgx import *

ngx = RsNgx('TCPIP::10.102.52.45::INSTR')
# Greetings, stranger...
print(f'Hello, I am: {ngx.utilities.idn_string}')
ngx.utilities.reset()

# Master switch for all the outputs - switch OFF
#   OUTPut:GENeral:STATe OFF
ngx.output.general.set_state(False)


# Select and set Output 1
#   INSTrument:SELect 1
ngx.instrument.select.set(1)
#   SOURce:VOLTage:LEVel:IMMediate:AMPlitude 3.3
ngx.source.voltage.level.immediate.set_amplitude(3.3)
#   SOURce:CURRent:LEVel:IMMediate:AMPlitude 0.1
ngx.source.current.level.immediate.set_amplitude(0.1)
# Prepare for setting the output to ON with the master switch
#   OUTPut:SELect ON
ngx.output.set_select(True)

# Select and set Output 2
#   INSTrument:SELect 2
ngx.instrument.select.set(2)
#   SOURce:VOLTage:LEVel:IMMediate:AMPlitude 5.1
ngx.source.voltage.level.immediate.set_amplitude(5.1)
#   SOURce:CURRent:LEVel:IMMediate:AMPlitude 0.05
ngx.source.current.level.immediate.set_amplitude(0.05)
# Prepare for setting the output to ON with the master switch
```

(continues on next page)

(continued from previous page)

```
#   OUTPut:SELect ON
ngx.output.set_select(True)

# The outputs are still OFF, they wait for this master switch:
#   OUTPut:GENeral:STATe ON
ngx.output.general.set_state(True)
# Insert a small pause to allow the instrument to settle the output
time.sleep(0.5)

#   INSTrument:SELect 1
ngx.instrument.select.set(1)
#                 READ?
measurement = ngx.read()
print(f'Measured values Output 1: {measurement.Voltage} V, {measurement.Current} A')

#   INSTrument:SELect 2
ngx.instrument.select.set(2)
#                 READ?
measurement = ngx.read()
print(f'Measured values Output 2: {measurement.Voltage} V, {measurement.Current} A')
ngx.close()
```

```
"""RsNgx example showing how to use channel persistence feature."""

import time
from RsNgx import *

RsNgx.assert_minimum_version('3.0.0.38')
ngx = RsNgx('TCPIP::10.102.52.45::INSTR')
print(f'Hello, I am: {ngx.utilities.idn_string}')
print(f'My installed options: {ngx.utilities.instrument_options}')
ngx.utilities.reset()

# Master switch for all the outputs - switch OFF
ngx.output.general.set_state(False)

# We clone the ngx object to ngx_ch1 and switch channel persistence to output channel 1
# It means, the driver makes sure that every command you sent with the ngx_ch1 goes to␣
↪output channel 1
ngx_ch1 = ngx.clone()
ngx_ch1.set_persistent_channel(1)

# Now we do the same for output channel 2
ngx_ch2 = ngx.clone()
ngx_ch2.set_persistent_channel(2)

# We can now use the ngx_ch1 and ngx_ch2 in any order, without having to call ngx.
↪instrument.select.set() in between:
ngx_ch1.source.voltage.level.immediate.set_amplitude(3.3)
ngx_ch2.source.voltage.level.immediate.set_amplitude(1.1)
ngx_ch1.source.current.level.immediate.set_amplitude(0.11)
ngx_ch2.source.current.level.immediate.set_amplitude(0.22)
```

(continues on next page)

(continued from previous page)

```
# Only Output 1 is ON
ngx_ch1.output.set_select(True)
# Output 2 stays OFF
ngx_ch2.output.set_select(False)

# For commands that are not channel - related, like for example reset() or Master switch␣
↪(below),
# you can use any of the objects: ngx, ngx_ch1, ngx_ch2
ngx.output.general.set_state(True)
# Insert a small pause to allow the instrument to settle the output
time.sleep(0.5)

# Read the measurement on both outputs:
measurement = ngx_ch1.read()
print(f'Measured values Output 1: {measurement.Voltage} V, {measurement.Current} A')
# Output 2 is not ON, we get NAN readings
measurement = ngx_ch2.read()
print(f'Measured values Output 2: {measurement.Voltage} V, {measurement.Current} A')

# In order to close the VISA session, you need to call the close() on the original␣
↪object 'ngx'
# Calling close() on the cloned objects 'ngx_ch1' or 'ngx_ch2' does not close the original␣
↪VISA session:
ngx_ch1.close()
print(f'I am alive and well and can call reset() ...')
ngx.utilities.reset()
ngx_ch2.close()
print(f'I can still switch the master OFF ...')
ngx.output.general.set_state(False)
ngx.close()
print(f'Finally, you destroyed me...')
```

```
"""RsNgx example showing how to make a screenshot of the instrument display."""

import time
from RsNgx import *

file_path = r'c:\temp\ngx_screenshot.png'
RsNgx.assert_minimum_version('3.0.0.38')
ngx = RsNgx('TCPIP::10.102.52.45::INSTR')
print(f'Hello, I am: {ngx.utilities.idn_string}')

ngx.display.window.text.set_data("My Greetings to you ...")
ngx.hardCopy.formatPy.set(enums.HcpyFormat.PNG)
picture = ngx.hardCopy.get_data()

file = open(file_path, 'wb')
file.write(picture)
file.close()
print(f'Screenshot saved to: {file_path}')

ngx.close()
```

CHAPTER

SIX

RSNGX API STRUCTURE

class RsNgx(*resource_name: str*, *id_query: bool = True*, *reset: bool = False*, *options: Optional[str] = None*, *direct_session: Optional[object] = None*)

289 total commands, 21 Subgroups, 1 group commands

Initializes new RsNgx session.

Parameter options tokens examples:

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

- `LoggingMode = On` - Sets the logging status right from the start. Default: `Off`
- `LoggingName = 'MyDevice'` - Sets the name to represent the session in the log entries. Default: `'resource_name'`
- `LogToGlobalTarget = True` - Sets the logging target to the class-property previously set with RsNgx.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 Measurement

Response structure. Fields:

- Voltage: float: No parameter help available
- Current: float: No parameter help available

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 RsNgx session.

classmethod from_existing_session(*session: object*, *options: Optional[str] = None*) → RsNgx

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

Parameters

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

classmethod get_global_logging_relative_timestamp() → datetime

Returns global common relative timestamp for log entries.

classmethod get_global_logging_target()

Returns global common target stream.

get_persistent_channel() → int

Returns the current persistent channel. If the persistence is switched off, the method returns 0.

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: Optional[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'

read() → Measurement

```
# SCPI: READ
value: Measurement = driver.read()
```

Queries for the next available readback for voltage and current of the selected channel.

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

reset_time_statistics() → None

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

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_persistent_channel(*channel: int*) → None

If set to 0 (default value), the driver behaves exactly as the SCPI interface - you can select the addressed channel with rsnrx.instrument.select.set(). If you set this property to 1 .. 8 (depending on how many

channels your device has), the drivers makes sure that for each function you call, this channel is preselected, no matter which channel is currently active. In this mode you can not use the rsnrx.instrument.select.set()

Subgroups

6.1 Apply

SCPI Commands

```
APPLy
```

class ApplyCls

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

set(*voltage: float*, *current: float*, *output: bool*) → None

```
# SCPI: APPLy
driver.apply.set(voltage = 1.0, current = 1.0, output = False)
```

Sets or queries the voltage and current value of the selected channel.

param voltage

- numeric: Numeric value for voltage in the range of 0.000 to 64.050.
- MIN | MINimum: Min voltage at 0.000 V.
- MAX | MAXimum: Max value for voltage at 64.050V.
- DEF | DEFault: Default voltage.

param current

- numeric: Numeric value for current in the range of 0.000 to 20.0100.
- MIN | MINimum: Min current at 0.000 A.
- MAX | MAXimum: Max value for current at 0.0100 A.
- DEF | DEFault: Numeric value for current.

param output

- OUT1 | OUTP1 | OUTPut1 | CH1: Selects output for channel 1.
- OUT2 | OUTP2 | OUTPut2 | CH2: Selects output for channel 2.
- OUT3 | OUTP3 | OUTPut3 | CH3: Selects output for channel 2.
- OUT4 | OUTP4 | OUTPut4 | CH4: Selects output for channel 4.

6.2 Arbitrary

SCPI Commands

```
ARBitrary:TRANsfer
ARBitrary:STATe
ARBitrary:DATA
ARBitrary:REPetitions
ARBitrary:CLEar
ARBitrary:SAVE
ARBitrary:LOAD
```

class ArbitraryCls

Arbitrary commands group definition. 25 total commands, 6 Subgroups, 7 group commands

clear() → None

```
# SCPI: ARBitrary:CLEar
driver.arbitrary.clear()
```

Clears the previous defined arbitrary waveform data for the selected channel.

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

```
# SCPI: ARBitrary:CLEar
driver.arbitrary.clear_with_opc()
```

Clears the previous defined arbitrary waveform data for the selected channel.

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

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

get_data() → List[float]

```
# SCPI: ARBitrary:DATA
value: List[float] = driver.arbitrary.get_data()
```

Sets or queries the arbitrary points for the previous selected channel. Max. 1024 arbitrary points can be defined. The dwell time between 2 arbitrary points is specified from 1 ms to 60 ms.

return
arg_0: No help available

get_repetitions() → int

```
# SCPI: ARBitrary:REPetitions
value: int = driver.arbitrary.get_repetitions()
```

Sets or queries the repetition rate of the defined arbitrary waveform for the previous selected channel. Up to 65535 repetitions are possible. If the repetition rate '0' is selected the arbitrary waveform of the previous selected channel is repeated infinitely.

return
arg_0: No help available

get_state() → bool

```
# SCPI: ARBitrary[:STATe]
value: bool = driver.arbitrary.get_state()
```

Sets or queries the QuickArb function for the previous selected channel.

return
arg_0: No help available

load() → None

```
# SCPI: ARBitrary:LOAD
driver.arbitrary.load()
```

Loads an arbitrary table from a file (filename specified with method RsNgx.Arbitrary.Fname.set)

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

```
# SCPI: ARBitrary:LOAD
driver.arbitrary.load_with_opc()
```

Loads an arbitrary table from a file (filename specified with method RsNgx.Arbitrary.Fname.set)

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

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

save() → None

```
# SCPI: ARBitrary:SAVE
driver.arbitrary.save()
```

Saves the current arbitrary table to a file (filename specified with method RsNgx.Arbitrary.Fname.set) .

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

```
# SCPI: ARBitrary:SAVE
driver.arbitrary.save_with_opc()
```

Saves the current arbitrary table to a file (filename specified with method RsNgx.Arbitrary.Fname.set) .

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

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

set_data(*arg_0: List[float]*) → None

```
# SCPI: ARBitrary:DATA
driver.arbitrary.set_data(arg_0 = [1.1, 2.2, 3.3])
```

Sets or queries the arbitrary points for the previous selected channel. Max. 1024 arbitrary points can be defined. The dwell time between 2 arbitrary points is specified from 1 ms to 60 ms.

param arg_0
Voltage and current settings depending on the instrument type. If the interpolation mode is sets to 1, it indicates that the mode is activated. If the interpolation mode is sets to 0, it indicates that the mode is not activated.

set_repetitions(*arg_0: int*) → None

```
# SCPI: ARBitrary:REPetitions
driver.arbitrary.set_repetitions(arg_0 = 1)
```

Sets or queries the repetition rate of the defined arbitrary waveform for the previous selected channel. Up to 65535 repetitions are possible. If the repetition rate '0' is selected the arbitrary waveform of the previous selected channel is repeated infinitely.

param arg_0
No help available

set_state(*arg_0: bool*) → None

```
# SCPI: ARBitrary[:STATe]
driver.arbitrary.set_state(arg_0 = False)
```

Sets or queries the QuickArb function for the previous selected channel.

param arg_0

- 1: QuickArb function is activated.
- 0: QuickArb function is deactivated.

set_transfer(*arg_0: int*) → None

```
# SCPI: ARBitrary:TRANsfer
driver.arbitrary.set_transfer(arg_0 = 1)
```

Transfers the defined arbitrary table.

param arg_0
1

Subgroups

6.2.1 Behavior

SCPI Commands

```
ARBitrary:BEHavior:END
```

class BehaviorCls

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

get_end() → ArbEndBehavior

```
# SCPI: ARBitrary:BEHavior:END
value: enums.ArbEndBehavior = driver.arbitrary.behavior.get_end()
```

Sets or queries the arbitrary endpoint behavior, when arbitrary function is finished.

return
arg_0: No help available

set_end(*arg_0: ArbEndBehavior*) → None

```
# SCPI: ARBitrary:BEHavior:END
driver.arbitrary.behavior.set_end(arg_0 = enums.ArbEndBehavior.HOLD)
```

Sets or queries the arbitrary endpoint behavior, when arbitrary function is finished.

param arg_0
HOLD | OFF OFF If the arbitrary function is finished, the respective channel is deactivated automatically. HOLD If the arbitrary function is finished, the last arbitrary point of the user-defined arbitrary list is held.

6.2.2 Block

SCPI Commands

```
ARBitrary:BLOCk:DATA
ARBitrary:BLOCk:REPetitions
ARBitrary:BLOCk:ENDPoint
ARBitrary:BLOCk:CLEar
ARBitrary:BLOCk
```

class BlockCls

Block commands group definition. 6 total commands, 1 Subgroups, 5 group commands

clear() → None

```
# SCPI: ARBitrary:BLOCk:CLEar
driver.arbitrary.block.clear()
```

Clears a file selected for the block under channel arbitrary settings. See also method RsNgx.Arbitrary.Block.value.

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

```
# SCPI: ARBitrary:BLOCk:CLEar
driver.arbitrary.block.clear_with_opc()
```

Clears a file selected for the block under channel arbitrary settings. See also method RsNgx.Arbitrary.Block.value.

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

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

get_data() → List[float]

```
# SCPI: ARBitrary:BLOCk:DATA
value: List[float] = driver.arbitrary.block.get_data()
```

Define the data points for a whole block.

return
arg_0: No help available

get_endpoint() → int

```
# SCPI: ARBitrary:BLOCk:ENDPoint
value: int = driver.arbitrary.block.get_endpoint()
```

Queries the number of data points of the block of arbitrary data.

return
result: No help available

get_repetitions() → int

```
# SCPI: ARBitrary:BLOCk:REPetitions
value: int = driver.arbitrary.block.get_repetitions()
```

Sets or queries the number of repetitions of the block of arbitrary data.

return
arg_0: No help available

get_value() → int

```
# SCPI: ARBitrary:BLOCk
value: int = driver.arbitrary.block.get_value()
```

Select individual block between 1 to 8 in an arbitrary sequence.

return
arg_0: No help available

set_data(*arg_0: List[float]*) → None

```
# SCPI: ARBitrary:BLOCk:DATA
driver.arbitrary.block.set_data(arg_0 = [1.1, 2.2, 3.3])
```

Define the data points for a whole block.

param arg_0
Voltage and current settings depending on the instrument type. If the interpolation mode is sets to 1, it indicates that the mode is activated. If the interpolation mode is sets to 0, it indicates that the mode is not activated.

set_repetitions(*arg_0: int*) → None

```
# SCPI: ARBitrary:BLOCk:REPetitions
driver.arbitrary.block.set_repetitions(arg_0 = 1)
```

Sets or queries the number of repetitions of the block of arbitrary data.

param arg_0
No help available

set_value(*arg_0: int*) → None

```
# SCPI: ARBitrary:BLOCk
driver.arbitrary.block.set_value(arg_0 = 1)
```

Select individual block between 1 to 8 in an arbitrary sequence.

param arg_0
No help available

Subgroups

6.2.2.1 Fname

SCPI Commands

```
ARBitrary:BLOCk:FNAMe
```

class FnameCls

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

get(*arg_0: str*, *arg_1: Optional[Filename] = None*) → str

```
# SCPI: ARBitrary:BLOCk:FNAMe
value: str = driver.arbitrary.block.fname.get(arg_0 = r1, arg_1 = enums.
↪Filename.DEF)
```

Sets or queries the filename for block of arbitrary data.

param arg_0
No help available

param arg_1
No help available

return
arg_0: No help available

set(*arg_0: str*, *arg_1: Optional[Filename] = None*) → None

```
# SCPI: ARBitrary:BLOCk:FNAMe
driver.arbitrary.block.fname.set(arg_0 = r1, arg_1 = enums.Filename.DEF)
```

Sets or queries the filename for block of arbitrary data.

param arg_0
No help available

param arg_1
No help available

6.2.3 Fname

SCPI Commands

```
ARBitrary:FNAMe
```

class FnameCls

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

get() → str

```
# SCPI: ARBitrary:FNAMe
value: str = driver.arbitrary.fname.get()
```

Sets or queries the file name and storage location for the QuickArb function.

return
filename: Filename of the QuickArb function.

set(*filename: str*, *file_location: Optional[Filename] = None*) → None

```
# SCPI: ARBitrary:FNAMe
driver.arbitrary.fname.set(filename = '1', file_location = enums.Filename.DEF)
```

Sets or queries the file name and storage location for the QuickArb function.

param filename
Filename of the QuickArb function.

param file_location

- INT: Internal memory
- EXT: USB stick
- DEF: Internal memory

6.2.4 Priority

SCPI Commands

```
ARBitrary:PRIority:MODE
```

class PriorityCls

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

get_mode() → PrioMode

```
# SCPI: ARBitrary:PRIority:MODE
value: enums.PrioMode = driver.arbitrary.priority.get_mode()
```

No command help available

return
arg_0: No help available

set_mode(*arg_0: PrioMode*) → None

```
# SCPI: ARBitrary:PRIority:MODE
driver.arbitrary.priority.set_mode(arg_0 = enums.PrioMode.CPM)
```

No command help available

param arg_0
No help available

6.2.5 Sequence

SCPI Commands

```
ARBitrary:SEQuence:REPetitions
ARBitrary:SEQuence:ENDPoint
ARBitrary:SEQuence:CLEar
```

class SequenceCls

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

clear() → None

```
# SCPI: ARBitrary:SEQuence:CLEar
driver.arbitrary.sequence.clear()
```

Clears the arbitrary sequence.

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

```
# SCPI: ARBitrary:SEQuence:CLEar
driver.arbitrary.sequence.clear_with_opc()
```

Clears the arbitrary sequence.

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

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

get_endpoint() → int

```
# SCPI: ARBitrary:SEQuence:ENDPoint
value: int = driver.arbitrary.sequence.get_endpoint()
```

Queries the total number of points of the arbitrary sequence.

return
result: No help available

get_repetitions() → int

```
# SCPI: ARBitrary:SEQuence:REPetitions
value: int = driver.arbitrary.sequence.get_repetitions()
```

Sets or queries the number of repetitions of the arbitrary sequence

return
arg_0: No help available

set_repetitions(*arg_0: int*) → None

```
# SCPI: ARBitrary:SEQuence:REPetitions
driver.arbitrary.sequence.set_repetitions(arg_0 = 1)
```

Sets or queries the number of repetitions of the arbitrary sequence

param arg_0
No help available

Subgroups

6.2.5.1 Behavior

SCPI Commands

```
ARBitrary:SEQuence:BEHavior:END
```

class BehaviorCls

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

get_end() → ArbEndBehavior

```
# SCPI: ARBitrary:SEQuence:BEHavior:END
value: enums.ArbEndBehavior = driver.arbitrary.sequence.behavior.get_end()
```

Sets or queries the arbitrary endpoint behavior, when QuickArb function is finished.

return
arg_0: No help available

set_end(*arg_0: ArbEndBehavior*) → None

```
# SCPI: ARBitrary:SEQuence:BEHavior:END
driver.arbitrary.sequence.behavior.set_end(arg_0 = enums.ArbEndBehavior.HOLD)
```

Sets or queries the arbitrary endpoint behavior, when QuickArb function is finished.

param arg_0

- OFF: If the QuickArb function is finished, the respective channel is deactivated automatically.
- HOLD: If the QuickArb function is finished, the last arbitrary point of the user-defined arbitrary list is held.

6.2.5.2 Transfer

SCPI Commands

```
ARBitrary:SEQuence:TRANsfer
```

class TransferCls

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

set() → None

```
# SCPI: ARBitrary:SEQuence:TRANsfer
driver.arbitrary.sequence.transfer.set()
```

Transfers the defined arbitrary table to the selected channel.

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

```
# SCPI: ARBitrary:SEQuence:TRANsfer
driver.arbitrary.sequence.transfer.set_with_opc()
```

Transfers the defined arbitrary table to the selected channel.

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

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

6.2.6 Triggered

SCPI Commands

```
ARBitrary:TRIGgered:STATe
ARBitrary:TRIGgered:MODE
```

class TriggeredCls

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

get_mode() → ArbTrigMode

```
# SCPI: ARBitrary:TRIGgered:MODE
value: enums.ArbTrigMode = driver.arbitrary.triggered.get_mode()
```

Sets or queries the arbitrary trigger mode of the previous selected channel.

return
arg_0: No help available

get_state() → int

```
# SCPI: ARBitrary:TRIGgered[:STATe]
value: int or bool = driver.arbitrary.triggered.get_state()
```

Sets or queries the trigger condition of the arbitrary for the selected channel.

return
arg_0: (integer or boolean) No help available

set_mode(*arg_0: ArbTrigMode*) → None

```
# SCPI: ARBitrary:TRIGgered:MODE
driver.arbitrary.triggered.set_mode(arg_0 = enums.ArbTrigMode.RUN)
```

Sets or queries the arbitrary trigger mode of the previous selected channel.

param arg_0
SINGle | RUN SINGle A trigger event starts only with one arbitrary sequence. RUN A trigger event starts the whole arbitrary sequences (with all repetitions) .

set_state(*arg_0: int*) → None

```
# SCPI: ARBitrary:TRIGgered[:STATe]
driver.arbitrary.triggered.set_state(arg_0 = 1)
```

Sets or queries the trigger condition of the arbitrary for the selected channel.

param arg_0
(integer or boolean) - OFF: There is no DIO pin that has a mode set to arbitrary for the selected channel. - 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8: DIO pin/s are enabled with a mode set to arbitrary for the selected channel.When DIO pin is enabled with arbitrary mode, QuickArb function of the channel assigned to that pin will be enabled when the correct voltage is applied to the DIO pin.

Subgroups

6.2.6.1 Group

SCPI Commands

```
ARBitrary:TRIGgered:GROup:STATe
```

class GroupCls

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

get_state() → int

```
# SCPI: ARBitrary:TRIGgered:GROup[:STATe]
value: int or bool = driver.arbitrary.triggered.group.get_state()
```

Sets or queries the trigger condition of the arbitrary step group for the selected channel.

return
arg_0: (integer or boolean) No help available

set_state(*arg_0: int*) → None

```
# SCPI: ARBitrary:TRIGgered:GROup[:STATe]
driver.arbitrary.triggered.group.set_state(arg_0 = 1)
```

Sets or queries the trigger condition of the arbitrary step group for the selected channel.

param arg_0
(integer or boolean) - OFF: There is no DIO pin that has a mode set to arbitrary step group for the selected channel. - 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8: DIO pin/s are enabled with a mode set to arbitrary step group for the selected channel.When DIO pin is enabled with arbitrary step point mode, QuickArb function will step to the next point when the correct voltage is applied to the DIO pin.

6.2.6.2 Point

SCPI Commands

```
ARBitrary:TRIGgered:POINt:STATe
```

class PointCls

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

get_state() → int

```
# SCPI: ARBitrary:TRIGgered:POINt[:STATe]
value: int or bool = driver.arbitrary.triggered.point.get_state()
```

Sets or queries the trigger condition of the arbitrary step point for the selected channel.

return
arg_0: (integer or boolean) No help available

set_state(*arg_0: int*) → None

```
# SCPI: ARBitrary:TRIGgered:POINt[:STATe]
driver.arbitrary.triggered.point.set_state(arg_0 = 1)
```

Sets or queries the trigger condition of the arbitrary step point for the selected channel.

param arg_0
(integer or boolean) - OFF: There is no DIO pin that has a mode set to arbitrary step point for the selected channel. - 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8: DIO pin/s are enabled with a mode set to arbitrary step point for the selected channel.When DIO pin is enabled with arbitrary step point mode, QuickArb function will step to the next point when the correct voltage is applied to the DIO pin.

6.3 Battery

SCPI Commands

```
BATTery:STATus
```

class BatteryCls

Battery commands group definition. 26 total commands, 2 Subgroups, 1 group commands

get_status() → str

```
# SCPI: BATTery:STATus
value: str = driver.battery.get_status()
```

Queries the status of the battery (idle, charging or discharging) .

return
result: No help available

Subgroups

6.3.1 Model

SCPI Commands

```
BATTery:MODel:SAVE
BATTery:MODel:LOAD
BATTery:MODel:TRANsfer
BATTery:MODel:CAPacity
BATTery:MODel:ISOC
BATTery:MODel:CLEar
```

class ModelCls

Model commands group definition. 11 total commands, 3 Subgroups, 6 group commands

clear() → None

```
# SCPI: BATTery:MODel:CLEar
driver.battery.model.clear()
```

Clears the current battery model.

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

```
# SCPI: BATTery:MODel:CLEar
driver.battery.model.clear_with_opc()
```

Clears the current battery model.

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

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

get_capacity() → float

```
# SCPI: BATTery:MODel:CAPacity
value: float = driver.battery.model.get_capacity()
```

Sets or queries the battery model capacity.

return
arg_0: Sets the battery model capacity.

get_isoc() → float

```
# SCPI: BATTery:MODel:ISOC
value: float = driver.battery.model.get_isoc()
```

Sets or queries the initial state of charge (SoC) of the battery model.

return
arg_0: No help available

load() → None

```
# SCPI: BATTery:MODel:LOAD
driver.battery.model.load()
```

Loads a battery model for editing.

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

```
# SCPI: BATTery:MODel:LOAD
driver.battery.model.load_with_opc()
```

Loads a battery model for editing.

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

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

save() → None

```
# SCPI: BATTery:MODel:SAVE
driver.battery.model.save()
```

Saves the current battery model to a file

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

```
# SCPI: BATTery:MODel:SAVE
driver.battery.model.save_with_opc()
```

Saves the current battery model to a file

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

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

set_capacity(*arg_0: float*) → None

```
# SCPI: BATTery:MODel:CAPacity
driver.battery.model.set_capacity(arg_0 = 1.0)
```

Sets or queries the battery model capacity.

param arg_0
Sets the battery model capacity.

set_isoc(*arg_0: float*) → None

```
# SCPI: BATTery:MODel:ISOC
driver.battery.model.set_isoc(arg_0 = 1.0)
```

Sets or queries the initial state of charge (SoC) of the battery model.

param arg_0
Initial state of charge (SoC) for the battery model.

set_transfer(*arg_0: int*) → None

```
# SCPI: BATTery:MODel:TRANsfer
driver.battery.model.set_transfer(arg_0 = 1)
```

Transfers the loaded battery model into the channel.

param arg_0
1 | 2

Subgroups

6.3.1.1 Current

class CurrentCls

Current commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Subgroups

6.3.1.1.1 Limit

SCPI Commands

```
BATTery:MODel:CURRent:LIMit:EOD
BATTery:MODel:CURRent:LIMit:REGular
BATTery:MODel:CURRent:LIMit:EOC
```

class LimitCls

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

get_eoc() → float

```
# SCPI: BATTery:MODel:CURRent:LIMit:EOC
value: float = driver.battery.model.current.limit.get_eoc()
```

Sets or queries the current limit of the battery model at end-of-charge.

return
arg_0: Sets the current limit of the battery model at end-of-charge.

get_eod() → float

```
# SCPI: BATTery:MODel:CURRent:LIMit:EOD
value: float = driver.battery.model.current.limit.get_eod()
```

Sets or queries the current limit of the battery model at end-of-discharge.

return
arg_0: Sets the current limit of the battery model at end-of-discharge.

get_regular() → float

```
# SCPI: BATTery:MODel:CURRent:LIMit:REGular
value: float = driver.battery.model.current.limit.get_regular()
```

Sets or queries the current limit of the battery model at regular charge level.

return
arg_0: No help available

`set_eoc`(*arg_0: float*) → None

```
# SCPI: BATTery:MODel:CURRent:LIMit:EOC
driver.battery.model.current.limit.set_eoc(arg_0 = 1.0)
```

Sets or queries the current limit of the battery model at end-of-charge.

param arg_0
Sets the current limit of the battery model at end-of-charge.

`set_eod`(*arg_0: float*) → None

```
# SCPI: BATTery:MODel:CURRent:LIMit:EOD
driver.battery.model.current.limit.set_eod(arg_0 = 1.0)
```

Sets or queries the current limit of the battery model at end-of-discharge.

param arg_0
Sets the current limit of the battery model at end-of-discharge.

`set_regular`(*arg_0: float*) → None

```
# SCPI: BATTery:MODel:CURRent:LIMit:REGular
driver.battery.model.current.limit.set_regular(arg_0 = 1.0)
```

Sets or queries the current limit of the battery model at regular charge level.

param arg_0
No help available

6.3.1.2 Data

SCPI Commands

```
BATTery:MODel:DATA
```

`class DataCls`

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

`class DataStruct`

Response structure. Fields:

- Arg_0: List[int]: Sets the value for battery state of charge (SoC) .
- Arg_1: List[float]: Sets the value for battery open-circuit voltage (Voc) .
- Arg_2: List[float]: Sets the value for battery internal resistance (ESR) .

`get`() → DataStruct

```
# SCPI: BATTery:MODel:DATA
value: DataStruct = driver.battery.model.data.get()
```

Sets or queries the battery model data.

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

set(*arg_0: List[int]*, *arg_1: List[float]*, *arg_2: List[float]*) → None

```
# SCPI: BATTery:MODel:DATA
driver.battery.model.data.set(arg_0 = [1, 2, 3], arg_1 = [1.1, 2.2, 3.3], arg_2␣
↪= [1.1, 2.2, 3.3])
```

Sets or queries the battery model data.

param arg_0
Sets the value for battery state of charge (SoC) .

param arg_1
Sets the value for battery open-circuit voltage (Voc) .

param arg_2
Sets the value for battery internal resistance (ESR) .

6.3.1.3 Fname

SCPI Commands

```
BATTery:MODel:FNAMe
```

class FnameCls

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

get(*arg_0: str*, *arg_1: Optional[Filename] = None*) → str

```
# SCPI: BATTery:MODel:FNAMe
value: str = driver.battery.model.fname.get(arg_0 = r1, arg_1 = enums.Filename.
↪DEF)
```

Sets or queries a filename for the battery model.

param arg_0
No help available

param arg_1
No help available

return
arg_0: No help available

set(*arg_0: str*, *arg_1: Optional[Filename] = None*) → None

```
# SCPI: BATTery:MODel:FNAMe
driver.battery.model.fname.set(arg_0 = r1, arg_1 = enums.Filename.DEF)
```

Sets or queries a filename for the battery model.

param arg_0
No help available

param arg_1
No help available

6.3.2 Simulator

SCPI Commands

```
BATTery:SIMulator:ENABle
BATTery:SIMulator:SOC
BATTery:SIMulator:RESistance
BATTery:SIMulator:TVOLtage
```

class SimulatorCls

Simulator commands group definition. 14 total commands, 3 Subgroups, 4 group commands

get_enable() → bool

```
# SCPI: BATTery:SIMulator[:ENABle]
value: bool = driver.battery.simulator.get_enable()
```

Sets or queries the battery simulator state.

return
arg_0: 1 Enables the battery simulator state. 0 Disables the battery simulator state.

get_resistance() → float

```
# SCPI: BATTery:SIMulator:RESistance
value: float = driver.battery.simulator.get_resistance()
```

Queries the battery simulator internal resistance (ESR) .

return
result: No help available

get_soc() → float

```
# SCPI: BATTery:SIMulator:SOC
value: float = driver.battery.simulator.get_soc()
```

Sets or queries the state of charge (SoC) of the battery simulator.

return
arg_0: Sets SoC values.

get_tvoltage() → float

```
# SCPI: BATTery:SIMulator:TVOLtage
value: float = driver.battery.simulator.get_tvoltage()
```

Queries the terminal voltage (Vt) .

return
result: No help available

set_enable(*arg_0: bool*) → None

```
# SCPI: BATTery:SIMulator[:ENABle]
driver.battery.simulator.set_enable(arg_0 = False)
```

Sets or queries the battery simulator state.

param arg_0
1 Enables the battery simulator state. 0 Disables the battery simulator state.

set_soc(*arg_0: float*) → None

```
# SCPI: BATTery:SIMulator:SOC
driver.battery.simulator.set_soc(arg_0 = 1.0)
```

Sets or queries the state of charge (SoC) of the battery simulator.

param arg_0
Sets SoC values.

Subgroups

6.3.2.1 Capacity

SCPI Commands

```
BATTery:SIMulator:CAPacity:LIMit
BATTery:SIMulator:CAPacity
```

class CapacityCls

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

get_limit() → float

```
# SCPI: BATTery:SIMulator:CAPacity:LIMit
value: float = driver.battery.simulator.capacity.get_limit()
```

Sets or queries the full battery capacity.

return
arg_0: Defines the full battery capacity.

get_value() → float

```
# SCPI: BATTery:SIMulator:CAPacity
value: float = driver.battery.simulator.capacity.get_value()
```

Queries the remaining battery capacity.

return
result: No help available

set_limit(*arg_0: float*) → None

```
# SCPI: BATTery:SIMulator:CAPacity:LIMit
driver.battery.simulator.capacity.set_limit(arg_0 = 1.0)
```

Sets or queries the full battery capacity.

param arg_0
Defines the full battery capacity.

6.3.2.2 Current

SCPI Commands

```
BATTery:SIMulator:CURRent
```

class CurrentCls

Current commands group definition. 5 total commands, 1 Subgroups, 1 group commands

get_value() → float

```
# SCPI: BATTery:SIMulator:CURRent
value: float = driver.battery.simulator.current.get_value()
```

Queries the current (A) of battery simulator.

return
result: No help available

Subgroups

6.3.2.2.1 Limit

SCPI Commands

```
BATTery:SIMulator:CURRent:LIMit:EOD
BATTery:SIMulator:CURRent:LIMit:REGular
BATTery:SIMulator:CURRent:LIMit:EOC
BATTery:SIMulator:CURRent:LIMit
```

class LimitCls

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

get_eoc() → float

```
# SCPI: BATTery:SIMulator:CURRent:LIMit:EOC
value: float = driver.battery.simulator.current.limit.get_eoc()
```

Sets or queries the current limit at end-of-charge.

return
arg_0: Sets the current limit at end-of-charge.

get_eod() → float

```
# SCPI: BATTery:SIMulator:CURRent:LIMit:EOD
value: float = driver.battery.simulator.current.limit.get_eod()
```

Sets or queries the current limit at end-of-discharge.

return
arg_0: Sets the current limit at end-of-discharge.

get_regular() → float

```
# SCPI: BATTery:SIMulator:CURRent:LIMit:REGular
value: float = driver.battery.simulator.current.limit.get_regular()
```

Sets or queries the current limit at regular charge level.

return
arg_0: Sets the current limit at regular charge level.

get_value() → float

```
# SCPI: BATTery:SIMulator:CURRent:LIMit
value: float = driver.battery.simulator.current.limit.get_value()
```

Sets or queries the current limit from specific channel.

return
result: No help available

set_eoc(*arg_0: float*) → None

```
# SCPI: BATTery:SIMulator:CURRent:LIMit:EOC
driver.battery.simulator.current.limit.set_eoc(arg_0 = 1.0)
```

Sets or queries the current limit at end-of-charge.

param arg_0
Sets the current limit at end-of-charge.

set_eod(*arg_0: float*) → None

```
# SCPI: BATTery:SIMulator:CURRent:LIMit:EOD
driver.battery.simulator.current.limit.set_eod(arg_0 = 1.0)
```

Sets or queries the current limit at end-of-discharge.

param arg_0
Sets the current limit at end-of-discharge.

set_regular(*arg_0: float*) → None

```
# SCPI: BATTery:SIMulator:CURRent:LIMit:REGular
driver.battery.simulator.current.limit.set_regular(arg_0 = 1.0)
```

Sets or queries the current limit at regular charge level.

param arg_0
Sets the current limit at regular charge level.

6.3.2.3 Voc

SCPI Commands

```
BATTery:SIMulator:VOC:FULL
BATTery:SIMulator:VOC:EMPTy
BATTery:SIMulator:VOC
```

class VocCls

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

get_empty() → float

```
# SCPI: BATTery:SIMulator:VOC:EMPTy
value: float = driver.battery.simulator.voc.get_empty()
```

Queries the open circuit voltage (Voc) for empty SoC, i.e SoC = 0 %.

return
result: No help available

get_full() → float

```
# SCPI: BATTery:SIMulator:VOC:FULL
value: float = driver.battery.simulator.voc.get_full()
```

Queries the open circuit voltage (Voc) for full SoC, i.e SoC = 100 %.

return
result: No help available

get_value() → float

```
# SCPI: BATTery:SIMulator:VOC
value: float = driver.battery.simulator.voc.get_value()
```

Queries the open circuit voltage (Voc) .

return
result: No help available

6.4 Calibration

SCPI Commands

```
CALibration:DATE
CALibration:TYPE
CALibration:VALue
CALibration:USER
CALibration:SAVE
CALibration:TEMPerature
CALibration:COUNt
```

class CalibrationCls

Calibration commands group definition. 26 total commands, 7 Subgroups, 7 group commands

class DateStruct

Structure for reading output parameters. Fields:

- Arg_0: float: No parameter help available
- Arg_1: float: No parameter help available
- Arg_2: float: No parameter help available

get_count() → float

```
# SCPI: CALibration:COUNt
value: float = driver.calibration.get_count()
```

Queries the number of counts channel adjustment performed successfully.

return
result: No help available

get_date() → DateStruct

```
# SCPI: CALibration:DATE
value: DateStruct = driver.calibration.get_date()
```

Returns the channel adjustment date.

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

get_temperature() → float

```
# SCPI: CALibration:TEMPerature
value: float = driver.calibration.get_temperature()
```

Returns the temperature of selected channel.

return
result: No help available

get_type_py() → CalibrationType

```
# SCPI: CALibration:TYPE
value: enums.CalibrationType = driver.calibration.get_type_py()
```

No command help available

return
arg_0: (enum or string) No help available

get_user() → bool

```
# SCPI: CALibration:USER
value: bool = driver.calibration.get_user()
```

Starts the channel adjustment process.

return
arg_0: No help available

get_value() → float

```
# SCPI: CALibration:VALue
value: float = driver.calibration.get_value()
```

No command help available

return
arg_0: No help available

save() → None

```
# SCPI: CALibration:SAVE
driver.calibration.save()
```

Saves the channel adjustment.

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

```
# SCPI: CALibration:SAVE
driver.calibration.save_with_opc()
```

Saves the channel adjustment.

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

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

set_type_py(*arg_0: CalibrationType*) → None

```
# SCPI: CALibration:TYPE
driver.calibration.set_type_py(arg_0 = enums.CalibrationType.CURR)
```

No command help available

param arg_0
(enum or string) No help available

set_user(*arg_0: bool*) → None

```
# SCPI: CALibration:USER
driver.calibration.set_user(arg_0 = False)
```

Starts the channel adjustment process.

param arg_0
No help available

set_value(*arg_0: float*) → None

```
# SCPI: CALibration:VALue
driver.calibration.set_value(arg_0 = 1.0)
```

No command help available

param arg_0
No help available

Subgroups

6.4.1 Ainput

SCPI Commands

```
CALibration:AINPut:SAVE
CALibration:AINPut:DATA
CALibration:AINPut:DATE
CALibration:AINPut:COUNt
```

class AinputCls

Ainput commands group definition. 9 total commands, 5 Subgroups, 4 group commands

get_count() → int

```
# SCPI: CALibration:AINPut:COUNt
value: int = driver.calibration.ainput.get_count()
```

Queries the number of counts performed for analog input adjustment .

return
result: No help available

get_data() → float

```
# SCPI: CALibration:AINPut:DATA
value: float = driver.calibration.ainput.get_data()
```

Sets the analog input adjustment data.

return
arg_0: No help available

get_date() → str

```
# SCPI: CALibration:AINPut:DATE
value: str = driver.calibration.ainput.get_date()
```

Returns the analog input adjustment date ('DD-MM-YY') .

return
result: No help available

save() → None

```
# SCPI: CALibration:AINPut:SAVE
driver.calibration.ainput.save()
```

Saves the analog input adjustment.

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

```
# SCPI: CALibration:AINPut:SAVE
driver.calibration.ainput.save_with_opc()
```

Saves the analog input adjustment.

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

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

set_data(*arg_0: float*) → None

```
# SCPI: CALibration:AINPut:DATA
driver.calibration.ainput.set_data(arg_0 = 1.0)
```

Sets the analog input adjustment data.

param arg_0
Measured value from DMM.

Subgroups

6.4.1.1 Cancel

SCPI Commands

```
CALibration:AINPut:CANCel
```

class CancelCls

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

set() → None

```
# SCPI: CALibration:AINPut:CANCel
driver.calibration.ainput.cancel.set()
```

Cancels the analog input adjustment.

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

```
# SCPI: CALibration:AINPut:CANCel
driver.calibration.ainput.cancel.set_with_opc()
```

Cancels the analog input adjustment.

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

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

6.4.1.2 End

SCPI Commands

```
CALibration:AINPut:END
```

class EndCls

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

set() → None

```
# SCPI: CALibration:AINPut:END
driver.calibration.ainput.end.set()
```

Ends the analog input adjustment.

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

```
# SCPI: CALibration:AINPut:END
driver.calibration.ainput.end.set_with_opc()
```

Ends the analog input adjustment.

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

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

6.4.1.3 Start

SCPI Commands

```
CALibration:AINPut:STARt
```

class StartCls

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

get() → int

```
# SCPI: CALibration:AINPut:STARt
value: int = driver.calibration.ainput.start.get()
```

Selects the analog input pin for adjustment.

return
arg_0: No help available

set(*arg_0: int*) → None

```
# SCPI: CALibration:AINPut:STARt
driver.calibration.ainput.start.set(arg_0 = 1)
```

Selects the analog input pin for adjustment.

param arg_0
Input pin for adjustment.

6.4.1.4 Umax

SCPI Commands

```
CALibration:AINPut:UMAX
```

class UmaxCls

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

set() → None

```
# SCPI: CALibration:AINPut:UMAX
driver.calibration.ainput.umax.set()
```

Sets output voltage to high value 100 % of Vmax for analog input pin during adjustment.

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

```
# SCPI: CALibration:AINPut:UMAX
driver.calibration.ainput.umax.set_with_opc()
```

Sets output voltage to high value 100 % of Vmax for analog input pin during adjustment.

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

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

6.4.1.5 Umin

SCPI Commands

```
CALibration:AINPut:UMIN
```

class UminCls

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

set() → None

```
# SCPI: CALibration:AINPut:UMIN
driver.calibration.ainput.umin.set()
```

Sets the output voltage to low value 1 % of Vmax for analog input pin during adjustment.

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

```
# SCPI: CALibration:AINPut:UMIN
driver.calibration.ainput.umin.set_with_opc()
```

Sets the output voltage to low value 1 % of Vmax for analog input pin during adjustment.

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

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

6.4.2 Cancel

SCPI Commands

```
CALibration:CANCel
```

class CancelCls

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

set() → None

```
# SCPI: CALibration:CANCel
driver.calibration.cancel.set()
```

Cancels the channel adjustment.

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

```
# SCPI: CALibration:CANCel
driver.calibration.cancel.set_with_opc()
```

Cancels the channel adjustment.

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

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

6.4.3 Current

SCPI Commands

```
CALibration:CURRent:DATA
```

class CurrentCls

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

get_data() → float

```
# SCPI: CALibration:CURRent:DATA
value: float = driver.calibration.current.get_data()
```

Set the DMM reading after setting the output current level in channel adjustment process.

return
arg_0: No help available

set_data(*arg_0: float*) → None

```
# SCPI: CALibration:CURRent:DATA
driver.calibration.current.set_data(arg_0 = 1.0)
```

Set the DMM reading after setting the output current level in channel adjustment process.

param arg_0
Measured value from DMM.

Subgroups

6.4.3.1 Imax

SCPI Commands

```
CALibration:CURRent:IMAX
```

class ImaxCls

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

set() → None

```
# SCPI: CALibration:CURRent:IMAX
driver.calibration.current.imax.set()
```

Sets the output current to high value 100 % of Imax during current adjustment.

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

```
# SCPI: CALibration:CURRent:IMAX
driver.calibration.current.imax.set_with_opc()
```

Sets the output current to high value 100 % of Imax during current adjustment.

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

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

6.4.3.2 Imin

SCPI Commands

```
CALibration:CURRent:IMIN
```

class IminCls

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

set() → None

```
# SCPI: CALibration:CURRent:IMIN
driver.calibration.current.imin.set()
```

Sets the output current to low value 1 % of Imax during current adjustment.

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

```
# SCPI: CALibration:CURRent:IMIN
driver.calibration.current.imin.set_with_opc()
```

Sets the output current to low value 1 % of Imax during current adjustment.

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

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

6.4.4 End

SCPI Commands

```
CALibration:END
```

class EndCls

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

set() → None

```
# SCPI: CALibration:END
driver.calibration.end.set()
```

Ends the channel adjustment.

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

```
# SCPI: CALibration:END
driver.calibration.end.set_with_opc()
```

Ends the channel adjustment.

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

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

6.4.5 Point

SCPI Commands

```
CALibration:POINt
```

class PointCls

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

get(*arg_0: int*) → int

```
# SCPI: CALibration:POINt
value: int = driver.calibration.point.get(arg_0 = 1)
```

No command help available

param arg_0
No help available

return
arg_0: No help available

set(*arg_0: int*) → None

```
# SCPI: CALibration:POINt
driver.calibration.point.set(arg_0 = 1)
```

No command help available

param arg_0
No help available

6.4.6 Self

SCPI Commands

```
CALibration:SELF
```

class SelfCls

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

set() → None

```
# SCPI: CALibration:SELF
driver.calibration.self.set()
```

No command help available

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

```
# SCPI: CALibration:SELF
driver.calibration.self.set_with_opc()
```

No command help available

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

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

6.4.7 Voltage

SCPI Commands

```
CALibration:VOLTage:DATA
```

class VoltageCls

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

get_data() → float

```
# SCPI: CALibration:VOLTage:DATA
value: float = driver.calibration.voltage.get_data()
```

Sets the DMM reading after setting the output voltage level in channel adjustment process.

return
arg_0: No help available

set_data(*arg_0: float*) → None

```
# SCPI: CALibration:VOLTage:DATA
driver.calibration.voltage.set_data(arg_0 = 1.0)
```

Sets the DMM reading after setting the output voltage level in channel adjustment process.

param arg_0
Measured value from DMM.

Subgroups

6.4.7.1 Umax

SCPI Commands

```
CALibration:VOLTage:UMAX
```

class UmaxCls

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

set() → None

```
# SCPI: CALibration:VOLTage:UMAX
driver.calibration.voltage.umax.set()
```

Sets the output voltage to high value 100 % of Vmax during voltage adjustment.

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

```
# SCPI: CALibration:VOLTage:UMAX
driver.calibration.voltage.umax.set_with_opc()
```

Sets the output voltage to high value 100 % of Vmax during voltage adjustment.

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

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

6.4.7.2 Umin

SCPI Commands

```
CALibration:VOLTage:UMIN
```

class UminCls

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

set() → None

```
# SCPI: CALibration:VOLTage:UMIN
driver.calibration.voltage.umin.set()
```

Sets the output voltage to low value 1 % of Vmax during voltage adjustment.

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

```
# SCPI: CALibration:VOLTage:UMIN
driver.calibration.voltage.umin.set_with_opc()
```

Sets the output voltage to low value 1 % of Vmax during voltage adjustment.

Same as set, but waits for the operation to complete before continuing further. Use the RsNgx.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 Data

SCPI Commands

```
DATA:DELete
DATA:LIST
```

class DataCls

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

delete(*arg_0: str*) → None

```
# SCPI: DATA:DELete
driver.data.delete(arg_0 = '1')
```

Deletes the specified file from memory.

param arg_0
Filepath of the file.

get_list_py() → List[str]

```
# SCPI: DATA:LIST
value: List[str] = driver.data.get_list_py()
```

Queries all files in internal memory (‘/int/’) and external memory (‘/USB’) .

return
result: No help available

Subgroups

6.5.1 Data

SCPI Commands

```
DATA:DATA
```

class DataCls

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

get(*arg_0: str*) → str

```
# SCPI: DATA:DATA
value: str = driver.data.data.get(arg_0 = '1')
```

Returns the logging file data of the selected file. If manual trigger mode (trigger via TRIG function) is used, the logging function has to be activated. Without activating the logging function in the manual trigger mode, the instrument is not able to save a logging file internally or on the USB stick.

param arg_0
Filepath of the logging file data.

return
result: No help available

6.5.2 Points

SCPI Commands

```
DATA:POINts
```

class PointsCls

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

get(*arg_0: str*) → int

```
# SCPI: DATA:POINts
value: int = driver.data.points.get(arg_0 = r1)
```

Queries the number of measurements from the selected logging file. If manual trigger mode (trigger via TRIG function) is used, the logging function has to be activated. Without activating the logging function in the manual trigger mode, the instrument is not able to save a logging file internally or on the USB stick.

param arg_0
Filepath of the logging file data.

return
result: No help available

6.6 Dio

class DioCls

Dio commands group definition. 7 total commands, 2 Subgroups, 0 group commands

Subgroups

6.6.1 Fault

SCPI Commands

```
DIO:FAULt:STATe
DIO:FAULt:CHANnel
DIO:FAULt:SOURce
DIO:FAULt:SIGNal
```

class FaultCls

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

get_channel() → int

```
# SCPI: DIO:FAULt:CHANnel
value: int = driver.dio.fault.get_channel()
```

Sets or queries channel selection for the digital output fault source. See 'operation modes' in Figure 'Overview of trigger IO system'.

return
arg_0: 1 | 2

get_signal() → DioSignal

```
# SCPI: DIO:FAULt:SIGNal
value: enums.DioSignal = driver.dio.fault.get_signal()
```

Select digital output fault signal type.

return
arg_0: CONStant | PULSe CONStant An constant level trigger signal is sent out PULSe An output pulse of 100 ms trigger signal is sent out

get_source() → DioFaultSource

```
# SCPI: DIO:FAULt:SOURce
value: enums.DioFaultSource = driver.dio.fault.get_source()
```

Sets or queries the 'operation modes' of the digital output fault source See 'operation modes' in Figure 'Overview of trigger IO system'.

return
arg_0: CC | CV | CR | SINK | PROTection | OUTPut If 'OUTPut' is selected, the 'fault output' will be active if the output of the selected channel is off.

get_state() → bool

```
# SCPI: DIO:FAULt[:STATe]
value: bool = driver.dio.fault.get_state()
```

Sets or queries the digital output fault. See 'operation modes' in Figure 'Overview of trigger IO system'

return
arg_0: 1 Enables digital output fault. 0 Disables digital output fault.

set_channel(*arg_0: int*) → None

```
# SCPI: DIO:FAULt:CHANnel
driver.dio.fault.set_channel(arg_0 = 1)
```

Sets or queries channel selection for the digital output fault source. See 'operation modes' in Figure 'Overview of trigger IO system'.

param arg_0
1 | 2

set_signal(*arg_0: DioSignal*) → None

```
# SCPI: DIO:FAULt:SIGNal
driver.dio.fault.set_signal(arg_0 = enums.DioSignal.CONStant)
```

Select digital output fault signal type.

param arg_0
CONStant | PULSe CONStant An constant level trigger signal is sent out PULSe An output pulse of 100 ms trigger signal is sent out

set_source(*arg_0: DioFaultSource*) → None

```
# SCPI: DIO:FAULt:SOURce
driver.dio.fault.set_source(arg_0 = enums.DioFaultSource.CC)
```

Sets or queries the 'operation modes' of the digital output fault source See 'operation modes' in Figure 'Overview of trigger IO system'.

param arg_0
CC | CV | CR | SINK | PROTection | OUTPut If 'OUTPut' is selected, the 'fault output' will be active if the output of the selected channel is off.

set_state(*arg_0: bool*) → None

```
# SCPI: DIO:FAULt[:STATe]
driver.dio.fault.set_state(arg_0 = False)
```

Sets or queries the digital output fault. See 'operation modes' in Figure 'Overview of trigger IO system'

param arg_0
1 Enables digital output fault. 0 Disables digital output fault.

6.6.2 Output

class OutputCls

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

Subgroups

6.6.2.1 Signal

SCPI Commands

```
DIO:OUTPut:SIGNal
```

class SignalCls

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

get(*arg_0: int*) → DioSignal

```
# SCPI: DIO:OUTPut:SIGNal
value: enums.DioSignal = driver.dio.output.signal.get(arg_0 = 1)
```

Select digital output signal type.

param arg_0
Channel number.

return
arg_1: CONStant | PULSe CONStant An constant level trigger signal is sent out PULSe An output pulse of 100 ms trigger signal is sent out

set(*arg_0: int*, *arg_1: DioSignal*) → None

```
# SCPI: DIO:OUTPut:SIGNal
driver.dio.output.signal.set(arg_0 = 1, arg_1 = enums.DioSignal.CONStant)
```

Select digital output signal type.

param arg_0
Channel number.

param arg_1
CONStant | PULSe CONStant An constant level trigger signal is sent out PULSe An output pulse of 100 ms trigger signal is sent out

6.6.2.2 Source

SCPI Commands

```
DIO:OUTPut:SOURce
```

class SourceCls

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

get(*arg_0: int*) → DioOutSource

```
# SCPI: DIO:OUTPut:SOURce
value: enums.DioOutSource = driver.dio.output.source.get(arg_0 = 1)
```

Sets or queries the digital output source. See 'operation modes' in Figure 'Overview of trigger IO system'.

param arg_0
1 Channel selection for the digital output source.

return
arg_1: OUTPut | TRIGger | FORCed OUTPut Selected channel output is used as the digital output source. TRIGger Selected channel external trigger signal is used as the digital output source. FORCed Selected output is forced to high level and can be switched by method RsNgx.Dio.Output.State.set command.

set(*arg_0: int*, *arg_1: DioOutSource*) → None

```
# SCPI: DIO:OUTPut:SOURce
driver.dio.output.source.set(arg_0 = 1, arg_1 = enums.DioOutSource.FORCed)
```

Sets or queries the digital output source. See 'operation modes' in Figure 'Overview of trigger IO system'.

param arg_0
1 Channel selection for the digital output source.

param arg_1
OUTPut | TRIGger | FORCed OUTPut Selected channel output is used as the digital output source. TRIGger Selected channel external trigger signal is used as the digital output source. FORCed Selected output is forced to high level and can be switched by method RsNgx.Dio.Output.State.set command.

6.6.2.3 State

SCPI Commands

```
DIO:OUTPut:STATe
```

class StateCls

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

get(*arg_0: int*) → bool

```
# SCPI: DIO:OUTPut[:STATe]
value: bool = driver.dio.output.state.get(arg_0 = 1)
```

Sets or queries the digital output channel selection.

param arg_0
1 Channel selection for the digital output source.

return
arg_1: 1 | 0 Enables or disables the digital output state.

set(*arg_0: int*, *arg_1: bool*) → None

```
# SCPI: DIO:OUTPut[:STATe]
driver.dio.output.state.set(arg_0 = 1, arg_1 = False)
```

Sets or queries the digital output channel selection.

param arg_0
1 Channel selection for the digital output source.

param arg_1
1 | 0 Enables or disables the digital output state.

6.7 Display

SCPI Commands

```
DISPlay:BRIGhtness
```

class DisplayCls

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

get_brightness() → float

```
# SCPI: DISPlay:BRIGhtness
value: float = driver.display.get_brightness()
```

Sets or queries the display brightness.

return
display_brightness: No help available

set_brightness(*display_brightness: float*) → None

```
# SCPI: DISPlay:BRIGhtness
driver.display.set_brightness(display_brightness = 1.0)
```

Sets or queries the display brightness.

param display_brightness
Displays brightness for the instrument.

Subgroups

6.7.1 Window

class WindowCls

Window commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Subgroups

6.7.1.1 Text

SCPI Commands

```
DISPlay:WINDow:TEXT:DATA
DISPlay:WINDow:TEXT:CLEar
```

class TextCls

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

clear() → None

```
# SCPI: DISPlay[:WINDow]:TEXT:CLEar
driver.display.window.text.clear()
```

Clears the text message box on the front display.

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

```
# SCPI: DISPlay[:WINDow]:TEXT:CLEar
driver.display.window.text.clear_with_opc()
```

Clears the text message box on the front display.

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

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

set_data(*custom_message: str*) → None

```
# SCPI: DISPlay[:WINDow]:TEXT[:DATA]
driver.display.window.text.set_data(custom_message = r1)
```

Shows the text message box on the front display.

param custom_message
New value for text message box.

6.8 Flog

SCPI Commands

```
FLOG:STATe
FLOG:DATA
FLOG:TRIGgered
FLOG:SRATe
FLOG:STIMe
FLOG:TARGet
```

class FlogCls

Flog commands group definition. 8 total commands, 1 Subgroups, 6 group commands

get_data() → List[float]

```
# SCPI: FLOG:DATA
value: List[float] = driver.flog.get_data()
```

Queries FastLog data as a block. The block is returned in the binary format starting in the sequence of voltage followed by current measurements, i.e. V, I, V, I, ... The R&S NGU accepts the line message EOI and/or the ASCII character NL (0Ah) as an indication that data transmission has been completed The binary data stream must be concluded with EOI or NL or EOI followed by NL. If the data stream is not concluded with either EOI or NL, the R&S NGU will wait for additional data. In the case of a binary data transmission, the R&S NGU ignores the bit combination NL (0Ah) within the data stream. The binary data block has the following structure: #<LengthofLength><Length><block_data>

INTRO_CMD_HELP: Example: #234<block_data>

- <LengthofLength> specifies how many positions the subsequent length specification occupies (‘2’ in the example)
- <Length> specifies the number of subsequent bytes (‘34’ in the example)
- <binary block data> specifies the binary block data of the specified length

To configure fastlog for scpi target, see Example ‘Configuring fastlog for scpi target’.

return
result: No help available

get_state() → bool

```
# SCPI: FLOG[:STATe]
value: bool = driver.flog.get_state()
```

Sets or queries the FastLog state.

return
arg_0: 1 Enables the FastLog state. 0 Disables the FastLog state.

get_stime() → float

```
# SCPI: FLOG:STIMe
value: float = driver.flog.get_stime()
```

No command help available

return
arg_0: No help available

get_symbol_rate() → FastLogSampleRate

```
# SCPI: FLOG:SRATe
value: enums.FastLogSampleRate = driver.flog.get_symbol_rate()
```

Sets or queries the sample rate of the FastLog function.

return
arg_0: No help available

get_target() → FastLogTarget

```
# SCPI: FLOG:TARGet
value: enums.FastLogTarget = driver.flog.get_target()
```

Chose the target the data shall be written to.

return
arg_0: SCPI | USB SCPI Transfer data to SCPI client. The sum of all sample rates have to be smaller or equal to 500 kS/s. USB Saves data to a binary file which is saved to the direectory specified in the 'Target Folder'.

get_triggered() → bool

```
# SCPI: FLOG:TRIGgered
value: bool = driver.flog.get_triggered()
```

Sets or queries the triggered state of FastLog. See Figure 'Overview of trigger IO system'.

return
arg_0: 1 Activates the FastLog state. 0 Deactivates the FastLog state.

set_state(*arg_0: bool*) → None

```
# SCPI: FLOG[:STATe]
driver.flog.set_state(arg_0 = False)
```

Sets or queries the FastLog state.

param arg_0
1 Enables the FastLog state. 0 Disables the FastLog state.

set_stime(*arg_0: float*) → None

```
# SCPI: FLOG:STIMe
driver.flog.set_stime(arg_0 = 1.0)
```

No command help available

param arg_0
No help available

set_symbol_rate(*arg_0: FastLogSampleRate*) → None

```
# SCPI: FLOG:SRATe
driver.flog.set_symbol_rate(arg_0 = enums.FastLogSampleRate.S001k)
```

Sets or queries the sample rate of the FastLog function.

param arg_0
500K | 250K | 125K | 62K5 | 31K25 | 15K625 | 7K812 | 3K906 | 1K953 | 976 | 488 | 244 | 122 | 61 | 30 | 15

set_target(*arg_0: FastLogTarget*) → None

```
# SCPI: FLOG:TARGet
driver.flog.set_target(arg_0 = enums.FastLogTarget.SCPI)
```

Chose the target the data shall be written to.

param arg_0
SCPI | USB SCPI Transfer data to SCPI client. The sum of all sample rates have to be smaller or equal to 500 kS/s. USB Saves data to a binary file which is saved to the direectory specified in the 'Target Folder'.

set_triggered(*arg_0: bool*) → None

```
# SCPI: FLOG:TRIGgered
driver.flog.set_triggered(arg_0 = False)
```

Sets or queries the triggered state of FastLog. See Figure 'Overview of trigger IO system'.

param arg_0
1 Activates the FastLog state. 0 Deactivates the FastLog state.

Subgroups

6.8.1 File

SCPI Commands

```
FLOG:FILE:TPARtition
FLOG:FILE:DURation
```

class FileCls

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

get_duration() → float

```
# SCPI: FLOG:FILE:DURation
value: float = driver.flog.file.get_duration()
```

Sets or queries the file write duration.

return
arg_0: No help available

get_tpartition() → str

```
# SCPI: FLOG:FILE:TPARtition
value: str = driver.flog.file.get_tpartition()
```

Selects the external partition to which the data is written into.

return
arg_0: Defines the external path partition to which the data is written in the USB stick.

set_duration(*arg_0: float*) → None

```
# SCPI: FLOG:FILE:DURation
driver.flog.file.set_duration(arg_0 = 1.0)
```

Sets or queries the file write duration.

param arg_0
Sets file write duration.

set_tpartition(*arg_0: str*) → None

```
# SCPI: FLOG:FILE:TPARtition
driver.flog.file.set_tpartition(arg_0 = r1)
```

Selects the external partition to which the data is written into.

param arg_0
Defines the external path partition to which the data is written in the USB stick.

6.9 Fuse

SCPI Commands

```
FUSE:STATe
FUSE:UNLink
```

class FuseCls

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

get_state() → bool

```
# SCPI: FUSE[:STATe]
value: bool = driver.fuse.get_state()
```

Sets or queries the state for over current protection (OCP) . See Example 'Configuring fuses'.

return
arg_0: - 1 | 0: - 1: Activates the OCP state. - 0: deactivates the OCP state.

set_state(*arg_0: bool*) → None

```
# SCPI: FUSE[:STATe]
driver.fuse.set_state(arg_0 = False)
```

Sets or queries the state for over current protection (OCP) . See Example 'Configuring fuses'.

param arg_0

- 1 | 0:
- 1: Activates the OCP state.
- 0: deactivates the OCP state.

set_unlink(*arg_0: List[int]*) → None

```
# SCPI: FUSE:UNLink
driver.fuse.set_unlink(arg_0 = [1, 2, 3])
```

Unlinks fuse linking from the other channels (Ch 1, Ch 2, Ch 3 or Ch 4) . See Example 'Configuring fuses'.

param arg_0
0 - Unlink all other channels to the previously selected channel.

Subgroups

6.9.1 Delay

SCPI Commands

```
FUSE:DELay:INITial
```

class DelayCls

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

get_initial() → float

```
# SCPI: FUSE:DELay:INITial
value: float = driver.fuse.delay.get_initial()
```

Sets the initial fuse delay time once output turns on.

return
voltage: No help available

set_initial(*voltage: float*) → None

```
# SCPI: FUSE:DELay:INITial
driver.fuse.delay.set_initial(voltage = 1.0)
```

Sets the initial fuse delay time once output turns on.

param voltage

- numeric value: Numeric value for initial fuse delay.
- MIN | MINimum: Min value for initial fuse delay.
- MAX | MAXimum: Max value for initial fuse delay.

6.9.2 Link

SCPI Commands

```
FUSE:LINK
```

class LinkCls

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

get(*arg_0: List[int]*) → List[int]

```
# SCPI: FUSE:LINK
value: List[int] = driver.fuse.link.get(arg_0 = [1, 2, 3])
```

Sets or queries the fuses of several selected channels (fuse linking) .

param arg_0
0 - Link all other channels to the previously selected channel.

return
arg_0: 0 - Link all other channels to the previously selected channel.

set(*arg_0: List[int]*) → None

```
# SCPI: FUSE:LINK
driver.fuse.link.set(arg_0 = [1, 2, 3])
```

Sets or queries the fuses of several selected channels (fuse linking) .

param arg_0
0 - Link all other channels to the previously selected channel.

6.9.3 Tripped

SCPI Commands

```
FUSE:TRIPped:CLEar
FUSE:TRIPped
```

class TrippedCls

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

clear() → None

```
# SCPI: FUSE:TRIPped:CLEar
driver.fuse.tripped.clear()
```

Resets the OCP state of the selected channel. If an OCP event has occurred before, the reset also erases the message on the display.

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

```
# SCPI: FUSE:TRIPped:CLEar
driver.fuse.tripped.clear_with_opc()
```

Resets the OCP state of the selected channel. If an OCP event has occurred before, the reset also erases the message on the display.

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

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

get_value() → bool

```
# SCPI: FUSE:TRIPped
value: bool = driver.fuse.tripped.get_value()
```

Queries the OCP state of the selected channel.

return
result: No help available

6.10 HardCopy

SCPI Commands

```
HCOPy:DATA
```

class HardCopyCls

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

get_data() → bytes

```
# SCPI: HCOPy:DATA
value: bytes = driver.hardCopy.get_data()
```

Returns the actual display content (screenshot) .

return
result: No help available

Subgroups

6.10.1 FormatPy

SCPI Commands

```
HCOPy:FORMat
```

class FormatPyCls

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

get(*format_py: HcpyFormat*) → HcpyFormat

```
# SCPI: HCOPy:FORMat
value: enums.HcpyFormat = driver.hardCopy.formatPy.get(format_py = enums.
↪HcpyFormat.BMP)
```

No command help available

param format_py
No help available

return
format_py: No help available

set(*format_py: HcpyFormat*) → None

```
# SCPI: HCOPy:FORMat
driver.hardCopy.formatPy.set(format_py = enums.HcpyFormat.BMP)
```

No command help available

param format_py
No help available

6.10.2 Size

SCPI Commands

```
HCOPy:SIZE:X
HCOPy:SIZE:Y
```

class SizeCls

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

get_x() → int

```
# SCPI: HCOPy:SIZE:X
value: int = driver.hardCopy.size.get_x()
```

Returns the horizontal dimension of the screenshots.

return
result: No help available

get_y() → int

```
# SCPI: HCOPy:SIZE:Y
value: int = driver.hardCopy.size.get_y()
```

Returns the vertical dimension of the screenshots.

return
result: No help available

6.11 Instrument

class InstrumentCls

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

Subgroups

6.11.1 Select

SCPI Commands

```
INSTrument:NSELect
```

class SelectCls

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

get() → int

```
# SCPI: INSTrument:NSELect
value: int = driver.instrument.select.get()
```

Selects or queries the channel by number.

return
channel: No help available

set(*channel: int*) → None

```
# SCPI: INSTrument:NSELect
driver.instrument.select.set(channel = 1)
```

Selects or queries the channel by number.

param channel
No help available

6.12 Interfaces

class InterfacesCls

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

Subgroups

6.12.1 Usb

SCPI Commands

```
INTerfaces:USB:CLASs
```

class UsbCls

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

get_class_py() → UsbClass

```
# SCPI: INTerfaces:USB:CLASs
value: enums.UsbClass = driver.interfaces.usb.get_class_py()
```

Sets or queries the USB class.

return
arg_0: No help available

set_class_py(*arg_0: UsbClass*) → None

```
# SCPI: INTerfaces:USB:CLASs
driver.interfaces.usb.set_class_py(arg_0 = enums.UsbClass.CDC)
```

Sets or queries the USB class.

param arg_0

- CDC: USB CDC connection.
- TMC: USB TMC connection.

6.13 Log

SCPI Commands

```
LOG:LOCation
LOG:DATA
LOG:TRIGgered
LOG:STATe
```

class LogCls

Log commands group definition. 11 total commands, 7 Subgroups, 4 group commands

get_data() → List[float]

```
# SCPI: LOG:DATA
value: List[float] = driver.log.get_data()
```

Returns 12 sets of latest logging data with minimum logging interval (8 ms) . Depending on the models, the data is returned in the following format for a 2-channel models: <Ch1_voltage>, <Ch1_current>,<Ch1_power>,<Ch2_voltage>, <Ch2_current>,<Ch2_power>, <Ch1_voltage>, <Ch1_current>,<Ch1_power>, <Ch2_voltage>, <Ch2_current>,<Ch2_power>...

return
result: No help available

get_location() → Filename

```
# SCPI: LOG:LOCation
value: enums.Filename = driver.log.get_location()
```

Sets or queries the logging location.

return
file_location: No help available

get_state() → bool

```
# SCPI: LOG[:STATe]
value: bool = driver.log.get_state()
```

Sets or queries the data logging state.

return
arg_0: No help available

get_triggered() → int

```
# SCPI: LOG:TRIGgered
value: int or bool = driver.log.get_triggered()
```

Sets or queries the state for manual trigger logging function.

return
arg_0: (integer or boolean) No help available

set_location(*file_location: Filename*) → None

```
# SCPI: LOG:LOCation
driver.log.set_location(file_location = enums.Filename.DEF)
```

Sets or queries the logging location.

param file_location
No help available

set_state(*arg_0: bool*) → None

```
# SCPI: LOG[:STATe]
driver.log.set_state(arg_0 = False)
```

Sets or queries the data logging state.

param arg_0

- 1: Data logging function is enabled.
- 0: Data logging function is disabled.

set_triggered(*arg_0: int*) → None

```
# SCPI: LOG:TRIGgered
driver.log.set_triggered(arg_0 = 1)
```

Sets or queries the state for manual trigger logging function.

param arg_0
(integer or boolean) 0 Manual trigger function is disabled. 1 Manual trigger function is enabled.

Subgroups

6.13.1 Channel

SCPI Commands

```
LOG:CHANnel:STATe
```

class ChannelCls

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

get_state() → bool

```
# SCPI: LOG:CHANnel[:STATe]
value: bool = driver.log.channel.get_state()
```

No command help available

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: LOG:CHANnel[:STATe]
driver.log.channel.set_state(arg_0 = False)
```

No command help available

param arg_0
No help available

6.13.2 Count

SCPI Commands

```
LOG:COUNt
```

class CountCls

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

get(*return_min_or_max: Optional[MinOrMax] = None*) → int

```
# SCPI: LOG:COUNt
value: int = driver.log.count.get(return_min_or_max = enums.MinOrMax.MAX)
```

Sets or queries the number of measurement values to be captured.

param return_min_or_max
No help available

return
result: No help available

set(*set_new_value: float*, *return_min_or_max: Optional[MinOrMax] = None*) → None

```
# SCPI: LOG:COUNt
driver.log.count.set(set_new_value = 1.0, return_min_or_max = enums.MinOrMax.
↪MAX)
```

Sets or queries the number of measurement values to be captured.

param set_new_value
No help available

param return_min_or_max
No help available

6.13.3 Duration

SCPI Commands

```
LOG:DURation
```

class DurationCls

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

get(*return_min_or_max: Optional[MinOrMax] = None*) → int

```
# SCPI: LOG:DURation
value: int = driver.log.duration.get(return_min_or_max = enums.MinOrMax.MAX)
```

Sets or queries the duration of the data logging.

param return_min_or_max
Returns the duration of the data logging.

return
result: No help available

set(*set_new_value: float*, *return_min_or_max: Optional[MinOrMax] = None*) → None

```
# SCPI: LOG:DURation
driver.log.duration.set(set_new_value = 1.0, return_min_or_max = enums.MinOrMax.
↪MAX)
```

Sets or queries the duration of the data logging.

param set_new_value

- numeric value: Duration of the data logging captured in the range of 0 s to 3.49*10^5 s.
- MIN | MINimum: Minimum duration of the data logging captured at 0 s.
- MAX | MAXimum: Maximum duration of the data logging captured at 3.49*10^5 s.

param return_min_or_max
Returns the duration of the data logging.

6.13.4 Fname

SCPI Commands

```
LOG:FNAMe
```

class FnameCls

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

get() → str

```
# SCPI: LOG:FNAMe
value: str = driver.log.fname.get()
```

Sets or queries the filename and storage location for the data logging.

return
set_new_value: No help available

set(*set_new_value: str*) → None

```
# SCPI: LOG:FNAMe
driver.log.fname.set(set_new_value = '1')
```

Sets or queries the filename and storage location for the data logging.

param set_new_value
No help available

6.13.5 Interval

SCPI Commands

```
LOG:INTerval
```

class IntervalCls

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

get(*return_min_or_max: Optional[MinOrMax] = None*) → float

```
# SCPI: LOG:INTerval
value: float = driver.log.interval.get(return_min_or_max = enums.MinOrMax.MAX)
```

Sets or queries the data logging measurement interval. The measurement interval describes the time between the recorded measurements.

param return_min_or_max
Returns the measurement interval.

return
result: No help available

set(*set_new_value: float*, *return_min_or_max: Optional[MinOrMax] = None*) → None

```
# SCPI: LOG:INTerval
driver.log.interval.set(set_new_value = 1.0, return_min_or_max = enums.MinOrMax.
↪MAX)
```

Sets or queries the data logging measurement interval. The measurement interval describes the time between the recorded measurements.

param set_new_value

- numeric value: Measurement interval in the range of 0.008 s to 600 s.
- MIN | MINimum: Minimum measurement interval is set at 0.008 s.
- MAX | MAXimum: Maximum measurement interval is set at 600 s.

param return_min_or_max
Returns the measurement interval.

6.13.6 Mode

SCPI Commands

```
LOG:MODE
```

class ModeCls

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

get(*arg_0: LogMode*) → LogMode

```
# SCPI: LOG:MODE
value: enums.LogMode = driver.log.mode.get(arg_0 = enums.LogMode.COUNt)
```

Sets or queries the data logging mode.

param arg_0

- UNLimited: Infinite data capture.
- COUNt: Number of measurement values to be captured.
- DURation: Duration of the measurement values capture.
- SPAN: Interval of the measurement values capture.

return
arg_0: No help available

set(*arg_0: LogMode*) → None

```
# SCPI: LOG:MODE
driver.log.mode.set(arg_0 = enums.LogMode.COUNt)
```

Sets or queries the data logging mode.

param arg_0

- UNLimited: Infinite data capture.
- COUNt: Number of measurement values to be captured.
- DURation: Duration of the measurement values capture.
- SPAN: Interval of the measurement values capture.

6.13.7 Stime

SCPI Commands

```
LOG:STIMe
```

class StimeCls

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

class SetStruct

Structure for setting input parameters. Fields:

- Year: int: No parameter help available
- Month: int: No parameter help available
- Day: int: No parameter help available
- Hour: int: No parameter help available
- Minute: int: No parameter help available
- Second: int: No parameter help available

get() → str

```
# SCPI: LOG:STIMe
value: str = driver.log.stime.get()
```

Sets or queries the start time of the data logging function.

return
result: No help available

set(*structure: SetStruct*) → None

```
# SCPI: LOG:STIMe
structure = driver.log.stime.SetStruct()
structure.Year: int = 1
structure.Month: int = 1
structure.Day: int = 1
structure.Hour: int = 1
structure.Minute: int = 1
structure.Second: int = 1
driver.log.stime.set(structure)
```

Sets or queries the start time of the data logging function.

param structure
for set value, see the help for SetStruct structure arguments.

6.14 Measure

class MeasureCls

Measure commands group definition. 21 total commands, 2 Subgroups, 0 group commands

Subgroups

6.14.1 Scalar

class ScalarCls

Scalar commands group definition. 20 total commands, 5 Subgroups, 0 group commands

Subgroups

6.14.1.1 Current

class CurrentCls

Current commands group definition. 5 total commands, 1 Subgroups, 0 group commands

Subgroups

6.14.1.1.1 Dc

SCPI Commands

```
MEASure:SCALar:CURRent:DC:MAX
MEASure:SCALar:CURRent:DC:AVG
MEASure:SCALar:CURRent:DC:MIN
MEASure:SCALar:CURRent:DC:STATistic
MEASure:SCALar:CURRent:DC
```

class DcCls

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

get_avg() → float

```
# SCPI: MEASure[:SCALar]:CURRent[:DC]:AVG
value: float = driver.measure.scalar.current.dc.get_avg()
```

Queries the average measured output current.

return
result: No help available

get_max() → float

```
# SCPI: MEASure[:SCALar]:CURRent[:DC]:MAX
value: float = driver.measure.scalar.current.dc.get_max()
```

Queries the maximum measured output current.

return
result: No help available

get_min() → float

```
# SCPI: MEASure[:SCALar]:CURRent[:DC]:MIN
value: float = driver.measure.scalar.current.dc.get_min()
```

Queries the minimum measured output power.

return
result: No help available

get_statistic() → float

```
# SCPI: MEASure[:SCALar]:CURRent[:DC]:STATistic
value: float = driver.measure.scalar.current.dc.get_statistic()
```

Queries the current statistics of the selected channel

return
result: No help available

get_value() → float

```
# SCPI: MEASure[:SCALar]:CURRent[:DC]
value: float = driver.measure.scalar.current.dc.get_value()
```

Queries the currently measured current of the selected channel.

return
result: No help available

6.14.1.2 Energy

SCPI Commands

```
MEASure:SCALar:ENERgy:STATe
MEASure:SCALar:ENERgy:RESet
MEASure:SCALar:ENERgy
```

class EnergyCls

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

get_state() → bool

```
# SCPI: MEASure[:SCALar]:ENERgy:STATe
value: bool = driver.measure.scalar.energy.get_state()
```

Sets or queries the energy counter state for the selected channel.

return
arg_0: No help available

get_value() → float

```
# SCPI: MEASure[:SCALar]:ENERgy
value: float = driver.measure.scalar.energy.get_value()
```

Queries the measured the current released energy value of the previous selected channel.

return
result: No help available

reset() → None

```
# SCPI: MEASure[:SCALar]:ENERgy:RESet
driver.measure.scalar.energy.reset()
```

Resets the energy counter for the selected channel.

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

```
# SCPI: MEASure[:SCALar]:ENERgy:RESet
driver.measure.scalar.energy.reset_with_opc()
```

Resets the energy counter for the selected channel.

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

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

set_state(*arg_0: bool*) → None

```
# SCPI: MEASure[:SCALar]:ENERgy:STATe
driver.measure.scalar.energy.set_state(arg_0 = False)
```

Sets or queries the energy counter state for the selected channel.

param arg_0

- 1: Activates the energy counter.
- 0: Deactivates the energy counter.

6.14.1.3 Power

SCPI Commands

```
MEASure:SCALar:POWer:MAX
MEASure:SCALar:POWer:AVG
MEASure:SCALar:POWer:MIN
MEASure:SCALar:POWer:STATistic
MEASure:SCALar:POWer
```

class PowerCls

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

get_avg() → float

```
# SCPI: MEASure[:SCALar]:POWer:AVG
value: float = driver.measure.scalar.power.get_avg()
```

Queries the average measured output power.

return
result: No help available

get_max() → float

```
# SCPI: MEASure[:SCALar]:POWer:MAX
value: float = driver.measure.scalar.power.get_max()
```

Queries the maximum measured output power.

return
result: No help available

get_min() → float

```
# SCPI: MEASure[:SCALar]:POWer:MIN
value: float = driver.measure.scalar.power.get_min()
```

Queries the minimum measured output power.

return
result: No help available

get_statistic() → float

```
# SCPI: MEASure[:SCALar]:POWer:STATistic
value: float = driver.measure.scalar.power.get_statistic()
```

Queries the power statistics of the selected channel.

return
result: No help available

get_value() → float

```
# SCPI: MEASure[:SCALar]:POWer
value: float = driver.measure.scalar.power.get_value()
```

Queries the currently emitted power of the selected channel

return
result: No help available

6.14.1.4 Statistic

SCPI Commands

```
MEASure:SCALar:STATistic:RESet
MEASure:SCALar:STATistic:COUNt
```

class StatisticCls

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

get_count() → int

```
# SCPI: MEASure[:SCALar]:STATistic:COUNt
value: int = driver.measure.scalar.statistic.get_count()
```

Returns the number of samples measured in the statistics for voltage/current/power

return
result: No help available

reset() → None

```
# SCPI: MEASure[:SCALar]:STATistic:RESet
driver.measure.scalar.statistic.reset()
```

Resets the minimum, maximum and average statistic values for voltage, current, and power. Additionally this command resets the measured energy.

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

```
# SCPI: MEASure[:SCALar]:STATistic:RESet
driver.measure.scalar.statistic.reset_with_opc()
```

Resets the minimum, maximum and average statistic values for voltage, current, and power. Additionally this command resets the measured energy.

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

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

6.14.1.5 Voltage

class VoltageCls

Voltage commands group definition. 5 total commands, 1 Subgroups, 0 group commands

Subgroups

6.14.1.5.1 Dc

SCPI Commands

```
MEASure:SCALar:VOLTage:DC:MAX
MEASure:SCALar:VOLTage:DC:AVG
MEASure:SCALar:VOLTage:DC:MIN
MEASure:SCALar:VOLTage:DC:STATistic
MEASure:SCALar:VOLTage:DC
```

class DcCls

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

get_avg() → float

```
# SCPI: MEASure[:SCALar][:VOLTage][:DC]:AVG
value: float = driver.measure.scalar.voltage.dc.get_avg()
```

Queries the average measured output voltage.

return
result: No help available

get_max() → float

```
# SCPI: MEASure[:SCALar][:VOLTage][:DC]:MAX
value: float = driver.measure.scalar.voltage.dc.get_max()
```

Queries the maximum measured output voltage.

return
result: No help available

get_min() → float

```
# SCPI: MEASure[:SCALar][:VOLTage][:DC]:MIN
value: float = driver.measure.scalar.voltage.dc.get_min()
```

Queries the minimum measured output voltage.

return
result: No help available

get_statistic() → float

```
# SCPI: MEASure[:SCALar][:VOLTage][:DC]:STATistic
value: float = driver.measure.scalar.voltage.dc.get_statistic()
```

Queries the voltage statistics of the selected channel.

return
result: No help available

get_value() → float

```
# SCPI: MEASure[:SCALar][:VOLTage][:DC]
value: float = driver.measure.scalar.voltage.dc.get_value()
```

Queries the currently measured voltage of the selected channel.

return
result: No help available

6.14.2 Voltage

SCPI Commands

```
MEASure:VOLTage:DVM
```

class VoltageCls

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

get_dvm() → float

```
# SCPI: MEASure:VOLTage:DVM
value: float = driver.measure.voltage.get_dvm()
```

Queries the voltmeter measurement (if DVM is enabled) . The DVM is available only with NGU201 model equipped with R&S NGU-K104 (P/N: 3663.0390.02) .

return
result: No help available

6.15 Output

SCPI Commands

```
OUTPut:FTResponse
OUTPut:STATe
OUTPut:SELect
```

class OutputCls

Output commands group definition. 13 total commands, 6 Subgroups, 3 group commands

get_ft_response() → bool

```
# SCPI: OUTPut:FTResponse
value: bool = driver.output.get_ft_response()
```

Sets or queries the fast transient response state.

return
arg_0: No help available

get_select() → bool

```
# SCPI: OUTPut:SELect
value: bool = driver.output.get_select()
```

Sets or queries the output state of selected channel.

return
arg_0: No help available

get_state() → bool

```
# SCPI: OUTPut[:STATe]
value: bool = driver.output.get_state()
```

Sets or queries the output state of the previous selected channels.

return
arg_0: No help available

set_ft_response(*arg_0: bool*) → None

```
# SCPI: OUTPut:FTResponse
driver.output.set_ft_response(arg_0 = False)
```

Sets or queries the fast transient response state.

param arg_0
No help available

set_select(*arg_0: bool*) → None

```
# SCPI: OUTPut:SELect
driver.output.set_select(arg_0 = False)
```

Sets or queries the output state of selected channel.

param arg_0

- 0: Deactivates the selected channel.
- 1: Activates the selected channel.

set_state(*arg_0: bool*) → None

```
# SCPI: OUTPut[:STATe]
driver.output.set_state(arg_0 = False)
```

Sets or queries the output state of the previous selected channels.

param arg_0

- 0: Switches off previous selected channels.
- 1: Switches on previous selected channels.

Subgroups

6.15.1 Delay

SCPI Commands

```
OUTPut:DELay:DURation
OUTPut:DELay:STATe
```

class DelayCls

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

get_duration() → float

```
# SCPI: OUTPut:DELay:DURation
value: float = driver.output.delay.get_duration()
```

Sets or queries the duration for output delay.

return
sequence_delay: No help available

get_state() → bool

```
# SCPI: OUTPut:DELay[:STATe]
value: bool = driver.output.delay.get_state()
```

Sets or queries the output delay state for the selected channel.

return
arg_0: No help available

set_duration(*sequence_delay: float*) → None

```
# SCPI: OUTPut:DELay:DURation
driver.output.delay.set_duration(sequence_delay = 1.0)
```

Sets or queries the duration for output delay.

param sequence_delay

- numeric value: Numeric value of the duration in seconds.
- MIN | MINimum: Minimum value of the duration at 0.001 seconds.
- MAX | MAXimum: Maximum value of the duration at 10.00 seconds.

set_state(*arg_0: bool*) → None

```
# SCPI: OUTPut:DELay[:STATe]
driver.output.delay.set_state(arg_0 = False)
```

Sets or queries the output delay state for the selected channel.

param arg_0

- 0: Deactivates output delay for the selected channel.
- 1: Activates output delay for the selected channel.

6.15.2 General

SCPI Commands

```
OUTPut:GENeral:STATe
```

class GeneralCls

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

get_state() → bool

```
# SCPI: OUTPut:GENeral[:STATe]
value: bool = driver.output.general.get_state()
```

Sets or queries all previous selected channels simultaneously

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: OUTPut:GENeral[:STATe]
driver.output.general.set_state(arg_0 = False)
```

Sets or queries all previous selected channels simultaneously

param arg_0

- 0: Switches off previous selected channels simultaneously.
- 1: Switches on previous selected channels simultaneously.

6.15.3 Impedance

SCPI Commands

```
OUTPut:IMPedance:STATe
OUTPut:IMPedance
```

class ImpedanceCls

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

get_state() → bool

```
# SCPI: OUTPut:IMPedance[:STATe]
value: bool = driver.output.impedance.get_state()
```

Sets or queries the output impedance state for the selected channel.

return
arg_0: No help available

get_value() → float

```
# SCPI: OUTPut:IMPedance
value: float = driver.output.impedance.get_value()
```

Sets or queries source impedance for the signal specified in ohms.

return
arg_0: numeric value | MIN | MINimum | MAX | MAXimum | DEF | DEFault | list numeric value Numeric value of the impedance ohm. MIN | MINimum Minimum value of the impedance at -0.05 ohms. MAX | MAXimum Maximum value of the impedance at 100 ohms. DEF Default value of the impedance at 0 ohms. Unit: ohm

set_state(*arg_0: bool*) → None

```
# SCPI: OUTPut:IMPedance[:STATe]
driver.output.impedance.set_state(arg_0 = False)
```

Sets or queries the output impedance state for the selected channel.

param arg_0

- 0: Deactivates output impedance for the selected channel.
- 1: Activates output impedance for the selected channel.

set_value(*arg_0: float*) → None

```
# SCPI: OUTPut:IMPedance
driver.output.impedance.set_value(arg_0 = 1.0)
```

Sets or queries source impedance for the signal specified in ohms.

param arg_0
numeric value | MIN | MINimum | MAX | MAXimum | DEF | DEFault | list numeric value Numeric value of the impedance ohm. MIN | MINimum Minimum value of the impedance at -0.05 ohms. MAX | MAXimum Maximum value of the impedance at 100 ohms. DEF Default value of the impedance at 0 ohms. Unit: ohm

6.15.4 Mode

SCPI Commands

```
OUTPut:MODE:CAPacitance
OUTPut:MODE
```

class ModeCls

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

get_capacitance() → str

```
# SCPI: OUTPut:MODE:CAPacitance
value: str = driver.output.mode.get_capacitance()
```

No command help available

return
arg_0: No help available

get_value() → str

```
# SCPI: OUTPut:MODE
value: str = driver.output.mode.get_value()
```

Sets or queries the output mode.

return
arg_0: AUTO | SINK | SOURce | list AUTO If operates in auto mode, the R&S NGU goes into sink or source mode depending on the voltage across the output terminal. If voltage across the output terminal exceeds the set voltage, current flows into the instrument, e.g. the instrument is now operating in sink mode; vv if voltage across output terminal is below set voltage, instrument operates as a source mode. SINK If operates in sink mode, current flows into the instrument. On display, current is shown as negative current. SOURce If operates in source mode, current flows out from the instrument.

set_capacitance(*arg_0: str*) → None

```
# SCPI: OUTPut:MODE:CAPacitance
driver.output.mode.set_capacitance(arg_0 = r1)
```

No command help available

param arg_0
No help available

set_value(*arg_0: str*) → None

```
# SCPI: OUTPut:MODE
driver.output.mode.set_value(arg_0 = r1)
```

Sets or queries the output mode.

param arg_0
AUTO | SINK | SOURce | list AUTO If operates in auto mode, the R&S NGU goes into sink or source mode depending on the voltage across the output terminal. If voltage

across the output terminal exceeds the set voltage, current flows into the instrument, e.g. the instrument is now operating in sink mode; vv if voltage across output terminal is below set voltage, instrument operates as a source mode. SINK If operates in sink mode, current flows into the instrument. On display, current is shown as negative current. SOURce If operates in source mode, current flows out from the instrument.

6.15.5 SymbolRate

SCPI Commands

```
OUTPut:SRATe:STATe
```

class SymbolRateCls

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

get_state() → bool

```
# SCPI: OUTPut:SRATe[:STATe]
value: bool = driver.output.symbolRate.get_state()
```

Sets or queries the reduce slew rate option for the selected channel.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: OUTPut:SRATe[:STATe]
driver.output.symbolRate.set_state(arg_0 = False)
```

Sets or queries the reduce slew rate option for the selected channel.

param arg_0

- 1: Activates reduce slew rate option for the selected channel.
- 0: Deactivates reduce slew rate option for the selected channel.

6.15.6 Triggered

class TriggeredCls

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

Subgroups

6.15.6.1 Behavior

SCPI Commands

```
OUTPut:TRIGgered:BEHavior
```

class BehaviorCls

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

get() → str

```
# SCPI: OUTPut:TRIGgered:BEHavior
value: str = driver.output.triggered.behavior.get()
```

Sets or queries output behavior when a trigger event occurs.

return
arg_0: ON | OFF | GATed | list ON Output is set on when a trigger event occurs. OFF Output is set off when a trigger event occurs. GATed Output is set gated when a trigger event occurs.

set(*arg_0: str*) → None

```
# SCPI: OUTPut:TRIGgered:BEHavior
driver.output.triggered.behavior.set(arg_0 = r1)
```

Sets or queries output behavior when a trigger event occurs.

param arg_0
ON | OFF | GATed | list ON Output is set on when a trigger event occurs. OFF Output is set off when a trigger event occurs. GATed Output is set gated when a trigger event occurs.

6.15.6.2 State

SCPI Commands

```
OUTPut:TRIGgered:STATe
```

class StateCls

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

get() → bool

```
# SCPI: OUTPut:TRIGgered[:STATe]
value: bool = driver.output.triggered.state.get()
```

Enables or disables the triggered event for output.

return
arg_0: 1 Trigger is enabled. 0 Trigger is disabled.

set(*arg_0: bool*) → None

```
# SCPI: OUTPut:TRIGgered[:STATe]
driver.output.triggered.state.set(arg_0 = False)
```

Enables or disables the triggered event for output.

param arg_0
1 Trigger is enabled. 0 Trigger is disabled.

6.16 Sense

class SenseCls

Sense commands group definition. 7 total commands, 3 Subgroups, 0 group commands

Subgroups

6.16.1 Current

class CurrentCls

Current commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Subgroups

6.16.1.1 Range

SCPI Commands

```
SENSe:CURRent:RANGe:UPPer
SENSe:CURRent:RANGe:LOWer
SENSe:CURRent:RANGe:AUTO
```

class RangeCls

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

get_auto() → bool

```
# SCPI: SENSe:CURRent:RANGe:AUTO
value: bool = driver.sense.current.range.get_auto()
```

Sets or queries auto range for current measurement accuracy.

return
arg_0: 1 Enables auto range for current. 0 Disables auto range for current.

get_lower() → float

```
# SCPI: SENSe:CURRent:RANGe:LOWer
value: float = driver.sense.current.range.get_lower()
```

No command help available

return
arg_0: No help available

get_upper() → float

```
# SCPI: SENSe:CURRent:RANGe[:UPPer]
value: float = driver.sense.current.range.get_upper()
```

Sets or queries the current range for measurement. There is a selection of 10 A, 1 A, 100 mA and 10 mA range.

return
arg_0: Defines the current range for measurement (10 A, 1 A, 100 mA and 10 mA) .
Unit: A

set_auto(*arg_0: bool*) → None

```
# SCPI: SENSe:CURRent:RANGe:AUTO
driver.sense.current.range.set_auto(arg_0 = False)
```

Sets or queries auto range for current measurement accuracy.

param arg_0
1 Enables auto range for current. 0 Disables auto range for current.

set_lower(*arg_0: float*) → None

```
# SCPI: SENSe:CURRent:RANGe:LOWer
driver.sense.current.range.set_lower(arg_0 = 1.0)
```

No command help available

param arg_0
No help available

set_upper(*arg_0: float*) → None

```
# SCPI: SENSe:CURRent:RANGe[:UPPer]
driver.sense.current.range.set_upper(arg_0 = 1.0)
```

Sets or queries the current range for measurement. There is a selection of 10 A, 1 A, 100 mA and 10 mA range.

param arg_0
Defines the current range for measurement (10 A, 1 A, 100 mA and 10 mA) . Unit: A

6.16.2 NplCycles

SCPI Commands

```
SENSe:NPLCycles
```

class NplCyclesCls

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

get(*arg_1: Optional[str] = None*) → float

```
# SCPI: [SENSe]:NPLCycles
value: float = driver.sense.nplCycles.get(arg_1 = r1)
```

Sets or queries the number of power line cycles for measurements.

param arg_1
list

return
arg_0: Number of power line cycles for measurements.

set(*arg_0: float*, *arg_1: Optional[str] = None*) → None

```
# SCPI: [SENSe]:NPLCycles
driver.sense.nplCycles.set(arg_0 = 1.0, arg_1 = r1)
```

Sets or queries the number of power line cycles for measurements.

param arg_0
Number of power line cycles for measurements.

param arg_1
list

6.16.3 Voltage

class VoltageCls

Voltage commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Subgroups

6.16.3.1 Range

SCPI Commands

```
SENSe:VOLTage:RANGe:UPPer
SENSe:VOLTage:RANGe:LOWer
SENSe:VOLTage:RANGe:AUTO
```

class RangeCls

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

get_auto() → bool

```
# SCPI: SENSe:VOLTage:RANGe:AUTO
value: bool = driver.sense.voltage.range.get_auto()
```

Sets or queries auto range for voltage measurement accuracy.

return
arg_0: 1 Enables auto range for voltage. 0 Disables auto range for voltage.

get_lower() → float

```
# SCPI: SENSe:VOLTage:RANGe:LOWer
value: float = driver.sense.voltage.range.get_lower()
```

No command help available

return
arg_0: No help available

get_upper() → float

```
# SCPI: SENSe:VOLTage:RANGe[:UPPer]
value: float = driver.sense.voltage.range.get_upper()
```

Sets or queries the voltage range for measurement. There is a selection of 20 V and 5 V range.

return
arg_0: Defines the voltage range for measurement (20 V and 5 V) . Unit: V

set_auto(*arg_0: bool*) → None

```
# SCPI: SENSe:VOLTage:RANGe:AUTO
driver.sense.voltage.range.set_auto(arg_0 = False)
```

Sets or queries auto range for voltage measurement accuracy.

param arg_0
1 Enables auto range for voltage. 0 Disables auto range for voltage.

set_lower(*arg_0: float*) → None

```
# SCPI: SENSe:VOLTage:RANGe:LOWer
driver.sense.voltage.range.set_lower(arg_0 = 1.0)
```

No command help available

param arg_0
No help available

set_upper(*arg_0: float*) → None

```
# SCPI: SENSe:VOLTage:RANGe[:UPPer]
driver.sense.voltage.range.set_upper(arg_0 = 1.0)
```

Sets or queries the voltage range for measurement. There is a selection of 20 V and 5 V range.

param arg_0
Defines the voltage range for measurement (20 V and 5 V) . Unit: V

6.17 Source

class SourceCls

Source commands group definition. 41 total commands, 8 Subgroups, 0 group commands

Subgroups

6.17.1 Alimit

SCPI Commands

```
SOURce:ALIMit:STATe
```

class AlimitCls

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

get_state() → bool

```
# SCPI: [SOURce]:ALIMit[:STATe]
value: bool = driver.source.alimit.get_state()
```

Sets or queries the safety limit state.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: [SOURce]:ALIMit[:STATe]
driver.source.alimit.set_state(arg_0 = False)
```

Sets or queries the safety limit state.

param arg_0

- 1: Activates the safety limit.
- 0: Deactivates the safety limit.

6.17.2 Current

SCPI Commands

```
SOURce:CURRent:RANGe
```

class CurrentCls

Current commands group definition. 13 total commands, 3 Subgroups, 1 group commands

get_range() → float

```
# SCPI: [SOURce]:CURRent:RANGe
value: float = driver.source.current.get_range()
```

No command help available

return
arg_0: No help available

set_range(*arg_0: float*) → None

```
# SCPI: [SOURce]:CURRent:RANGe
driver.source.current.set_range(arg_0 = 1.0)
```

No command help available

param arg_0
No help available

Subgroups

6.17.2.1 Level

class LevelCls

Level commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Subgroups

6.17.2.1.1 Immediate

SCPI Commands

```
SOURce:CURRent:LEVel:IMMediate:AMPLitude
```

class ImmediateCls

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

get_amplitude() → float

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate][:AMPLitude]
value: float = driver.source.current.level.immediate.get_amplitude()
```

Sets or queries the current value of the selected channel.

return

current: - numeric value: Numeric value in the range of 0.000 to 20.0100 . - MIN | MINimum: Minimum current at 0.0005 A. - MAX | MAXimum: Depending on the set voltage level, the maximum set current is 20.0100 A.For voltage range up to 32 V, maximum set current is 20.0100 A.For voltage range up to 64 V, maximum set current is 10.0100 A. - UP: Increases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement]. - DOWN: Decreases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement].

set_amplitude(*current: float*) → None

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate][:AMPLitude]
driver.source.current.level.immediate.set_amplitude(current = 1.0)
```

Sets or queries the current value of the selected channel.

param current

- numeric value: Numeric value in the range of 0.000 to 20.0100 .
- MIN | MINimum: Minimum current at 0.0005 A.
- MAX | MAXimum: Depending on the set voltage level, the maximum set current is 20.0100 A.For voltage range up to 32 V, maximum set current is 20.0100 A.For voltage range up to 64 V, maximum set current is 10.0100 A.
- UP: Increases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement].
- DOWN: Decreases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement].

Subgroups

6.17.2.1.1.1 Alimit

class AlimitCls

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

Subgroups

6.17.2.1.1.2 Lower

SCPI Commands

```
SOURce:CURRent:LEVel:IMMediate:ALIMit:LOWer
```

class LowerCls

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

get() → float

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate]:ALIMit:LOWer
value: float = driver.source.current.level.immediate.alimit.lower.get()
```

Sets or queries the lower safety limit for current.

return
result: No help available

set(*current: float*) → None

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate]:ALIMit:LOWer
driver.source.current.level.immediate.alimit.lower.set(current = 1.0)
```

Sets or queries the lower safety limit for current.

param current

- numeric value: Numeric value for lower safety limit.
- MIN | MINimum: Min value for lower safety limit.
- MAX | MAXimum: Max value for lower safety limit.

6.17.2.1.1.3 Upper

SCPI Commands

```
SOURce:CURRent:LEVel:IMMediate:ALIMit:UPPer
```

class UpperCls

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

get() → float

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate]:ALIMit[:UPPer]
value: float = driver.source.current.level.immediate.alimit.upper.get()
```

Sets or queries the upper safety limit for current.

return
result: No help available

set(*current: float*) → None

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate]:ALIMit[:UPPer]
driver.source.current.level.immediate.alimit.upper.set(current = 1.0)
```

Sets or queries the upper safety limit for current.

param current

- numeric value: Numeric value for upper safety limit.
- MIN | MINimum: Min value for upper safety limit.
- MAX | MAXimum: Max value for upper safety limit.

6.17.2.1.1.4 Step

class StepCls

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

Subgroups

6.17.2.1.1.5 Increment

SCPI Commands

```
SOURce:CURRent:LEVel:IMMediate:STEP:INCRement
```

class IncrementCls

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

get(*optional_default_step_query: Optional[DefaultStep] = None*) → float

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate]:STEP[:INCRement]
value: float = driver.source.current.level.immediate.step.increment.
↪get(optional_default_step_query = enums.DefaultStep.DEF)
```

Sets or queries the incremental step size for the [SOURce:]CURRent[:LEVel][:IMMediate][:AMPLitude] command.

param optional_default_step_query
Queries the default voltage step size.

return
result: No help available

set(*desired_stepsize: float*, *optional_default_step_query: Optional[DefaultStep] = None*) → None

```
# SCPI: [SOURce]:CURRent[:LEVel][:IMMediate]:STEP[:INCRement]
driver.source.current.level.immediate.step.increment.set(desired_stepsize = 1.0,
↪ optional_default_step_query = enums.DefaultStep.DEF)
```

Sets or queries the incremental step size for the [SOURce:]CURRent[:LEVel][:IMMediate][:AMPLitude] command.

param desired_stepsize

- numeric value: Step value in A.
- DEF | DEFault: Default value of stepsize.

param optional_default_step_query
Queries the default voltage step size.

6.17.2.2 Negative

class NegativeCls

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

Subgroups

6.17.2.2.1 Level

class LevelCls

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

Subgroups

6.17.2.2.1.1 Immediate

SCPI Commands

```
SOURce:CURRent:NEGative:LEVel:IMMediate:AMPLitude
```

class ImmediateCls

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

get_amplitude() → float

```
# SCPI: SOURce:CURRent:NEGative[:LEVel][:IMMediate][:AMPLitude]
value: float = driver.source.current.negative.level.immediate.get_amplitude()
```

Sets or queries the negative current value.

return
current: numeric value | MIN | MINimum | MAX | MAXimum | UP | DOWN | list numeric value Numeric value in the range of 0.000 to 20.0100 . MIN | MINimum Minimum current at 0.0005 A. MAX | MAXimum Depending on the set voltage level, the maximum set current is 20.0100 A. For voltage

range up to 32 V, maximum set current is 20.0100 A. For voltage range up to 64 V, maximum set current is 10.0100 A. UP Increases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement]. DOWN Decreases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement].

set_amplitude(*current: float*) → None

```
# SCPI: SOURce:CURRent:NEGative[:LEVel][:IMMediate][:AMPLitude]
driver.source.current.negative.level.immediate.set_amplitude(current = 1.0)
```

Sets or queries the negative current value.

param current
numeric value | MIN | MINimum | MAX | MAXimum | UP | DOWN | list numeric value Numeric value in the range of 0.000 to 20.0100 . MIN | MINimum Minimum current at 0.0005 A. MAX | MAXimum Depending on the set voltage level, the maximum set current is 20.0100 A. For voltage range up to 32 V, maximum set current is 20.0100 A. For voltage range up to 64 V, maximum set current is 10.0100 A. UP Increases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement]. DOWN Decreases current by a defined step size. See [SOURce:]CURRent[:LEVel][:IMMediate]:STEP[:INCRement].

6.17.2.3 Protection

SCPI Commands

```
SOURce:CURRent:PROTection:STATe
SOURce:CURRent:PROTection:UNLink
SOURce:CURRent:PROTection:TRIPped
SOURce:CURRent:PROTection:CLEar
```

class ProtectionCls

Protection commands group definition. 7 total commands, 2 Subgroups, 4 group commands

clear() → None

```
# SCPI: [SOURce]:CURRent:PROTection:CLEar
driver.source.current.protection.clear()
```

Resets the OCP state of the selected channel. If an OCP event has occurred before, the reset also erases the message on the display.

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

```
# SCPI: [SOURce]:CURRent:PROTection:CLEar
driver.source.current.protection.clear_with_opc()
```

Resets the OCP state of the selected channel. If an OCP event has occurred before, the reset also erases the message on the display.

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

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

`get_state()` → bool

```
# SCPI: [SOURce]:CURRent:PROTection[:STATe]
value: bool = driver.source.current.protection.get_state()
```

Sets or queries the OCP state.

return
arg_0: 1 Activates the OCP state. 0 deactivates the OCP state.

`get_tripped()` → bool

```
# SCPI: [SOURce]:CURRent:PROTection:TRIPped
value: bool = driver.source.current.protection.get_tripped()
```

Queries the OCP state of the selected channel.

return
result: No help available

`set_state`(*arg_0: bool*) → None

```
# SCPI: [SOURce]:CURRent:PROTection[:STATe]
driver.source.current.protection.set_state(arg_0 = False)
```

Sets or queries the OCP state.

param arg_0
1 Activates the OCP state. 0 deactivates the OCP state.

`set_unlink`(*arg_0: int*) → None

```
# SCPI: [SOURce]:CURRent:PROTection:UNLink
driver.source.current.protection.set_unlink(arg_0 = 1)
```

Unlink fuse linking from the other channel (Ch1 or Ch2) . See Example 'Configuring fuses'.

param arg_0
1 | 2

Subgroups

6.17.2.3.1 Delay

SCPI Commands

```
SOURce:CURRent:PROTection:DELay:INITial
SOURce:CURRent:PROTection:DELay
```

`class DelayCls`

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

get_initial() → float

```
# SCPI: [SOURce]:CURRent:PROTection:DELay:INITial
value: float = driver.source.current.protection.delay.get_initial()
```

Sets or queries the initial fuse delay time once output turns on.

return
voltage: No help available

get_value() → float

```
# SCPI: [SOURce]:CURRent:PROTection:DELay
value: float = driver.source.current.protection.delay.get_value()
```

Sets or queries the fuse delay time.

return
voltage: No help available

set_initial(*voltage: float*) → None

```
# SCPI: [SOURce]:CURRent:PROTection:DELay:INITial
driver.source.current.protection.delay.set_initial(voltage = 1.0)
```

Sets or queries the initial fuse delay time once output turns on.

param voltage
No help available

set_value(*voltage: float*) → None

```
# SCPI: [SOURce]:CURRent:PROTection:DELay
driver.source.current.protection.delay.set_value(voltage = 1.0)
```

Sets or queries the fuse delay time.

param voltage
No help available

6.17.2.3.2 Link

SCPI Commands

```
SOURce:CURRent:PROTection:LINK
```

class LinkCls

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

get(*arg_0: int*) → int

```
# SCPI: [SOURce]:CURRent:PROTection:LINK
value: int = driver.source.current.protection.link.get(arg_0 = 1)
```

Sets or queries the fuses of several selected channels (fuse linking) .

param arg_0
1 | 2

return
arg_0: 1 | 2

set(*arg_0: int*) → None

```
# SCPI: [SOURce]:CURRent:PROTection:LINK
driver.source.current.protection.link.set(arg_0 = 1)
```

Sets or queries the fuses of several selected channels (fuse linking) .

param arg_0
1 | 2

6.17.3 Modulation

SCPI Commands

```
SOURce:MODulation
```

class ModulationCls

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

get(*arg_0: bool*, *arg_1: Optional[str] = None*) → bool

```
# SCPI: [SOURce]:MODulation
value: bool = driver.source.modulation.get(arg_0 = False, arg_1 = r1)
```

No command help available

param arg_0
No help available

param arg_1
No help available

return
arg_0: No help available

set(*arg_0: bool*, *arg_1: Optional[str] = None*) → None

```
# SCPI: [SOURce]:MODulation
driver.source.modulation.set(arg_0 = False, arg_1 = r1)
```

No command help available

param arg_0
No help available

param arg_1
No help available

Subgroups

6.17.3.1 Gain

SCPI Commands

```
SOURce:MODulation:GAIN
```

class GainCls

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

get(*arg_0: float*, *arg_1: Optional[str] = None*) → float

```
# SCPI: [SOURce]:MODulation:GAIN
value: float = driver.source.modulation.gain.get(arg_0 = 1.0, arg_1 = r1)
```

Sets or queries the modulation gain.

param arg_0
numeric value | MIN | MINimum | MAX | MAXimum | UP | DOWN | list numeric value Numeric value of the modulation gain. MIN | MINimum Minimum value of the modulation gain. MAX | MAXimum Maximum value of the modulation gain UP Increases gain by a defined step size. DOWN Decreases gain by a defined step size.

param arg_1
list

return
arg_0: numeric value | MIN | MINimum | MAX | MAXimum | UP | DOWN | list numeric value Numeric value of the modulation gain. MIN | MINimum Minimum value of the modulation gain. MAX | MAXimum Maximum value of the modulation gain UP Increases gain by a defined step size. DOWN Decreases gain by a defined step size.

set(*arg_0: float*, *arg_1: Optional[str] = None*) → None

```
# SCPI: [SOURce]:MODulation:GAIN
driver.source.modulation.gain.set(arg_0 = 1.0, arg_1 = r1)
```

Sets or queries the modulation gain.

param arg_0
numeric value | MIN | MINimum | MAX | MAXimum | UP | DOWN | list numeric value Numeric value of the modulation gain. MIN | MINimum Minimum value of the modulation gain. MAX | MAXimum Maximum value of the modulation gain UP Increases gain by a defined step size. DOWN Decreases gain by a defined step size.

param arg_1
list

6.17.4 Power

class PowerCls

Power commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Subgroups

6.17.4.1 Protection

SCPI Commands

```
SOURce:POWer:PROTection:STATe
SOURce:POWer:PROTection:LEVel
SOURce:POWer:PROTection:TRIPped
SOURce:POWer:PROTection:CLEar
```

class ProtectionCls

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

clear() → None

```
# SCPI: [SOURce]:POWer:PROTection:CLEar
driver.source.power.protection.clear()
```

Resets the OPP state of the selected channel. If an OPP event has occurred before, the reset also erases the message on the display.

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

```
# SCPI: [SOURce]:POWer:PROTection:CLEar
driver.source.power.protection.clear_with_opc()
```

Resets the OPP state of the selected channel. If an OPP event has occurred before, the reset also erases the message on the display.

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

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

get_level() → float

```
# SCPI: [SOURce]:POWer:PROTection:LEVel
value: float = driver.source.power.protection.get_level()
```

Sets or queries the overvoltage protection value of the selected channel.

return
voltage_protection: No help available

get_state() → bool

```
# SCPI: [SOURce]:POWer:PROTection[:STATe]
value: bool = driver.source.power.protection.get_state()
```

Sets or queries the OPP state of the previous selected channel.

return
arg_0: No help available

`get_tripped()` → bool

```
# SCPI: [SOURce]:POWer:PROTection:TRIPped
value: bool = driver.source.power.protection.get_tripped()
```

Queries the OPP state of the selected channel.

return
result: No help available

`set_level`(*voltage_protection: float*) → None

```
# SCPI: [SOURce]:POWer:PROTection:LEVel
driver.source.power.protection.set_level(voltage_protection = 1.0)
```

Sets or queries the overvoltage protection value of the selected channel.

param voltage_protection

- numeric value: Numeric value of the power protection level in watts.
- MIN | MINimum: Minimum value of the power protection level at 0.00 W.
- MAX | MAXimum: Maximum value of the power protection level at 200.00 W.
- DEF | DEFault: Default value of the power protection level at 200.00 W.

`set_state`(*arg_0: bool*) → None

```
# SCPI: [SOURce]:POWer:PROTection[:STATe]
driver.source.power.protection.set_state(arg_0 = False)
```

Sets or queries the OPP state of the previous selected channel.

param arg_0

- 0: OPP is deactivated
- 1: OPP is activated

6.17.5 Priority

SCPI Commands

```
SOURce:PRIority
```

`class PriorityCls`

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

`get`(*arg_0: str*, *channel_list: Optional[str] = None*) → str

```
# SCPI: SOURce:PRIority
value: str = driver.source.priority.get(arg_0 = r1, channel_list = r1)
```

Sets or queries the source priority mode.

param arg_0
VOLTage | CURRent | list VOLT Set the source measure unit to operate in voltage priority mode. CURR Set the source measure unit to operate in current priority mode.

param channel_list
list

return
arg_0: VOLTage | CURRent | list VOLT Set the source measure unit to operate in voltage priority mode. CURR Set the source measure unit to operate in current priority mode.

set(*arg_0: str*) → None

```
# SCPI: SOURce:PRIority
driver.source.priority.set(arg_0 = r1)
```

Sets or queries the source priority mode.

param arg_0
VOLTage | CURRent | list VOLT Set the source measure unit to operate in voltage priority mode. CURR Set the source measure unit to operate in current priority mode.

6.17.6 Protection

SCPI Commands

```
SOURce:PROTection:CLEar
```

class ProtectionCls

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

clear() → None

```
# SCPI: [SOURce]:PROTection:CLEar
driver.source.protection.clear()
```

Reset protection tripped state.

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

```
# SCPI: [SOURce]:PROTection:CLEar
driver.source.protection.clear_with_opc()
```

Reset protection tripped state.

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

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

6.17.7 Resistance

SCPI Commands

```
SOURce:RESistance:STATe
```

class ResistanceCls

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

get_state() → bool

```
# SCPI: [SOURce]:RESistance:STATe
value: bool = driver.source.resistance.get_state()
```

Sets or queries the constant resistance mode.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: [SOURce]:RESistance:STATe
driver.source.resistance.set_state(arg_0 = False)
```

Sets or queries the constant resistance mode.

param arg_0
No help available

Subgroups

6.17.7.1 Level

class LevelCls

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

Subgroups

6.17.7.1.1 Immediate

SCPI Commands

```
SOURce:RESistance:LEVel:IMMediate:AMPLitude
```

class ImmediateCls

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

get_amplitude() → float

```
# SCPI: [SOURce]:RESistance[:LEVel][:IMMediate][:AMPLitude]
value: float = driver.source.resistance.level.immediate.get_amplitude()
```

Sets or queries the constant resistance target value.

return
current: No help available

set_amplitude(*current: float*) → None

```
# SCPI: [SOURce]:RESistance[:LEVel][:IMMediate][:AMPLitude]
driver.source.resistance.level.immediate.set_amplitude(current = 1.0)
```

Sets or queries the constant resistance target value.

param current
No help available

6.17.8 Voltage

SCPI Commands

```
SOURce:VOLTage:RANGe
```

class VoltageCls

Voltage commands group definition. 17 total commands, 7 Subgroups, 1 group commands

get_range() → float

```
# SCPI: [SOURce]:VOLTage:RANGe
value: float = driver.source.voltage.get_range()
```

No command help available

return
arg_0: No help available

set_range(*arg_0: float*) → None

```
# SCPI: [SOURce]:VOLTage:RANGe
driver.source.voltage.set_range(arg_0 = 1.0)
```

No command help available

param arg_0
No help available

Subgroups

6.17.8.1 Ainput

SCPI Commands

```
SOURce:VOLTage:AINPut:STATe
SOURce:VOLTage:AINPut:INPut
```

class AinputCls

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

get_input_py() → str

```
# SCPI: [SOURce]:VOLTage:AINPut:INPut
value: str = driver.source.voltage.ainput.get_input_py()
```

Sets or queries the analog input mode.

return
arg_0: No help available

get_state() → bool

```
# SCPI: [SOURce]:VOLTage:AINPut[:STATe]
value: bool = driver.source.voltage.ainput.get_state()
```

Enables or disables the analog input for the selected channel.

return
arg_0: - 1: Analog input for selected channel is enabled. - 0: Analog input for selected channel is disabled.

set_input_py(*arg_0: str*) → None

```
# SCPI: [SOURce]:VOLTage:AINPut:INPut
driver.source.voltage.ainput.set_input_py(arg_0 = r1)
```

Sets or queries the analog input mode.

param arg_0

- VOLT: Voltage mode.
- CURR: Current mode.

set_state(*arg_0: bool*) → None

```
# SCPI: [SOURce]:VOLTage:AINPut[:STATe]
driver.source.voltage.ainput.set_state(arg_0 = False)
```

Enables or disables the analog input for the selected channel.

param arg_0

- 1: Analog input for selected channel is enabled.
- 0: Analog input for selected channel is disabled.

Subgroups

6.17.8.1.1 Triggered

SCPI Commands

```
SOURce:VOLTage:AINPut:TRIGgered:STATe
```

class TriggeredCls

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

get_state() → int

```
# SCPI: [SOURce]:VOLTage:AINPut:TRIGgered[:STATe]
value: int or bool = driver.source.voltage.ainput.triggered.get_state()
```

Sets or queries the trigger condition of the analog input for the selected channel.

return
arg_0: (integer or boolean) No help available

set_state(*arg_0: int*) → None

```
# SCPI: [SOURce]:VOLTage:AINPut:TRIGgered[:STATe]
driver.source.voltage.ainput.triggered.set_state(arg_0 = 1)
```

Sets or queries the trigger condition of the analog input for the selected channel.

param arg_0
(integer or boolean) - OFF: There is no DIO pin that has a mode set to Analog In for the selected channel. - 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8: DIO pin/s are enabled with a mode set to Analog In for the selected channel.When DIO pin is enabled with Analog In mode, analog input of the channel assigned to that pin will be enabled when the correct voltage is applied to the DIO pin.

6.17.8.2 Dvm

SCPI Commands

```
SOURce:VOLTage:DVM:STATe
```

class DvmCls

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

get_state() → bool

```
# SCPI: [SOURce]:VOLTage:DVM[:STATe]
value: bool = driver.source.voltage.dvm.get_state()
```

Sets or queries digital voltmeter measurements.

return
arg_0: 1 Enables digital voltmeter measurement. 0 Disables digital voltmeter measurement.

set_state(*arg_0: bool*) → None

```
# SCPI: [SOURce]:VOLTage:DVM[:STATe]
driver.source.voltage.dvm.set_state(arg_0 = False)
```

Sets or queries digital voltmeter measurements.

param arg_0
1 Enables digital voltmeter measurement. 0 Disables digital voltmeter measurement.

6.17.8.3 Level

class LevelCls

Level commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Subgroups

6.17.8.3.1 Immediate

SCPI Commands

```
SOURce:VOLTage:LEVel:IMMediate:AMPLitude
```

class ImmediateCls

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

get_amplitude() → float

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate][:AMPLitude]
value: float = driver.source.voltage.level.immediate.get_amplitude()
```

Sets or queries the voltage value of the selected channel.

return

voltage: - numeric value: Numeric value in V. - MIN | MINimum: Minimum voltage at 0.000 V. - MAX | MAXimum: Maximum voltage at 64.050 V. - UP: Increases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement]. - DOWN: Decreases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement].

set_amplitude(*voltage: float*) → None

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate][:AMPLitude]
driver.source.voltage.level.immediate.set_amplitude(voltage = 1.0)
```

Sets or queries the voltage value of the selected channel.

param voltage

- numeric value: Numeric value in V.
- MIN | MINimum: Minimum voltage at 0.000 V.
- MAX | MAXimum: Maximum voltage at 64.050 V.
- UP: Increases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement].
- DOWN: Decreases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement].

Subgroups

6.17.8.3.1.1 Alimit

class AlimitCls

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

Subgroups

6.17.8.3.1.2 Lower

SCPI Commands

```
SOURce:VOLTage:LEVel:IMMediate:ALIMit:LOWer
```

class LowerCls

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

get() → float

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate]:ALIMit:LOWer
value: float = driver.source.voltage.level.immediate.alimit.lower.get()
```

Sets or queries the lower safety limit for voltage.

return
result: No help available

set(*voltage: float*) → None

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate]:ALIMit:LOWer
driver.source.voltage.level.immediate.alimit.lower.set(voltage = 1.0)
```

Sets or queries the lower safety limit for voltage.

param voltage

- numeric value: Numeric value for safety limit.
- MIN | MINimum: Min value for lower safety limit.
- MAX | MAXimum: Max value for lower safety limit.

6.17.8.3.1.3 Upper

SCPI Commands

```
SOURce:VOLTage:LEVel:IMMediate:ALIMit:UPPer
```

class UpperCls

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

get() → float

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate]:ALIMit[:UPPer]
value: float = driver.source.voltage.level.immediate.alimit.upper.get()
```

Sets or queries the upper safety limit for voltage.

return
result: No help available

set(*current: float*) → None

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate]:ALIMit[:UPPer]
driver.source.voltage.level.immediate.alimit.upper.set(current = 1.0)
```

Sets or queries the upper safety limit for voltage.

param current

- numeric value: Numeric value for upper safety limit.
- MIN | MINimum: Min value for upper safety limit.
- MAX | MAXimum: Max value for upper safety limit.

6.17.8.3.1.4 Step

class StepCls

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

Subgroups

6.17.8.3.1.5 Increment

SCPI Commands

```
SOURce:VOLTage:LEVel:IMMediate:STEP:INCRement
```

class IncrementCls

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

get(*optional_default_step_query: Optional[DefaultStep] = None*) → float

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate]:STEP[:INCRement]
value: float = driver.source.voltage.level.immediate.step.increment.
↪get(optional_default_step_query = enums.DefaultStep.DEF)
```

Sets or queries the incremental step size for the [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] command.

param optional_default_step_query
No help available

return
result: No help available

set(*desired_stepsize: float*, *optional_default_step_query: Optional[DefaultStep] = None*) → None

```
# SCPI: [SOURce]:VOLTage[:LEVel][:IMMediate]:STEP[:INCRement]
driver.source.voltage.level.immediate.step.increment.set(desired_stepsize = 1.0,
↪ optional_default_step_query = enums.DefaultStep.DEF)
```

Sets or queries the incremental step size for the [SOURce:]VOLTage[:LEVel][:IMMediate][:AMPLitude] command.

param desired_stepsize
No help available

param optional_default_step_query
No help available

6.17.8.4 Negative

class NegativeCls

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

Subgroups

6.17.8.4.1 Level

class LevelCls

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

Subgroups

6.17.8.4.1.1 Immediate

SCPI Commands

```
SOURce:VOLTage:NEGative:LEVel:IMMediate:AMPLitude
```

class ImmediateCls

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

get_amplitude() → float

```
# SCPI: SOURce:VOLTage:NEGative[:LEVel][:IMMediate][:AMPLitude]
value: float = driver.source.voltage.negative.level.immediate.get_amplitude()
```

Sets or queries the negative voltage value. Applicable only with NGU401 model.

return
voltage: numeric value | MIN | MINimum | MAX | MAXimum | UP | DOWN | list numeric value Numeric value in V. MIN | MINimum Minimum voltage at 0.000 V. MAX | MAXimum Maximum voltage at 64.050 V. UP Increases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement]. DOWN Decreases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement]. Range: 0.000 to 64.050

set_amplitude(*voltage: float*) → None

```
# SCPI: SOURce:VOLTage:NEGative[:LEVel][:IMMediate][:AMPLitude]
driver.source.voltage.negative.level.immediate.set_amplitude(voltage = 1.0)
```

Sets or queries the negative voltage value. Applicable only with NGU401 model.

param voltage
numeric value | MIN | MINimum | MAX | MAXimum | UP | DOWN | list numeric value Numeric value in V. MIN | MINimum Minimum voltage at 0.000 V. MAX | MAXimum Maximum voltage at 64.050 V. UP Increases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement]. DOWN Decreases voltage by a defined step size. See [SOURce:]VOLTage[:LEVel][:IMMediate]:STEP[:INCRement]. Range: 0.000 to 64.050

6.17.8.5 Protection

SCPI Commands

```
SOURce:VOLTage:PROTection:STATe
SOURce:VOLTage:PROTection:LEVel
SOURce:VOLTage:PROTection:TRIPped
SOURce:VOLTage:PROTection:CLEar
```

class ProtectionCls

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

clear() → None

```
# SCPI: [SOURce]:VOLTage:PROTection:CLEar
driver.source.voltage.protection.clear()
```

Resets the OVP state of the selected channel. If an OVP event has occurred before, the reset also erases the message on the display.

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

```
# SCPI: [SOURce]:VOLTage:PROTection:CLEar
driver.source.voltage.protection.clear_with_opc()
```

Resets the OVP state of the selected channel. If an OVP event has occurred before, the reset also erases the message on the display.

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

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

get_level() → float

```
# SCPI: [SOURce]:VOLTage:PROTection:LEVel
value: float = driver.source.voltage.protection.get_level()
```

Sets or queries the overvoltage protection value of the selected channel.

return
voltage_protection: No help available

get_state() → bool

```
# SCPI: [SOURce]:VOLTage:PROTection[:STATe]
value: bool = driver.source.voltage.protection.get_state()
```

Sets or queries the OVP state of the previous selected channel.

return
en: No help available

get_tripped() → bool

```
# SCPI: [SOURce]:VOLTage:PROTection:TRIPped
value: bool = driver.source.voltage.protection.get_tripped()
```

Queries the OVP state of the selected channel.

return
result: No help available

set_level(*voltage_protection: float*) → None

```
# SCPI: [SOURce]:VOLTage:PROTection:LEVel
driver.source.voltage.protection.set_level(voltage_protection = 1.0)
```

Sets or queries the overvoltage protection value of the selected channel.

param voltage_protection

- numeric value: Numeric value for the overvoltage protection value in V.
- MIN | MINimum: Minimum value for the overvoltage protection value at 0.000 V.
- MAX | MAXimum: Maximum value for the overvoltage protection value at 64.050 V.

set_state(*en: bool*) → None

```
# SCPI: [SOURce]:VOLTage:PROTection[:STATe]
driver.source.voltage.protection.set_state(en = False)
```

Sets or queries the OVP state of the previous selected channel.

param en

- 0: OVP is deactivated
- 1: OVP is activated

6.17.8.6 Ramp

SCPI Commands

```
SOURce:VOLTage:RAMP:STATe
SOURce:VOLTage:RAMP:DURation
```

class RampCls

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

get_duration() → float

```
# SCPI: [SOURce]:VOLTage:RAMP:DURation
value: float = driver.source.voltage.ramp.get_duration()
```

Sets or queries the duration of the voltage ramp.

return
voltage: No help available

get_state() → bool

```
# SCPI: [SOURce]:VOLTage:RAMP[:STATe]
value: bool = driver.source.voltage.ramp.get_state()
```

Sets or queries the state of ramp function for the previous selected channel.

return
arg_0: No help available

set_duration(*voltage: float*) → None

```
# SCPI: [SOURce]:VOLTage:RAMP:DURation
driver.source.voltage.ramp.set_duration(voltage = 1.0)
```

Sets or queries the duration of the voltage ramp.

param voltage

- numeric value: Duration of the ramp function in seconds.
- MIN | MINimum: Minimum duration of the ramp function at 0.00 s.
- MAX | MAXimum: Maximum duration of the ramp function at 60.00 s.
- DEF | DEFault: Default duration of the ramp function at 0.01 s.

set_state(*arg_0: bool*) → None

```
# SCPI: [SOURce]:VOLTage:RAMP[:STATe]
driver.source.voltage.ramp.set_state(arg_0 = False)
```

Sets or queries the state of ramp function for the previous selected channel.

param arg_0

- 0: EasyRamp function is deactivated.
- 1: EasyRamp function is activated.

6.17.8.7 Sense

SCPI Commands

```
SOURce:VOLTage:SENSe:SOURce
```

class SenseCls

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

get_source() → str

```
# SCPI: [SOURce]:VOLTage:SENSe[:SOURce]
value: str = driver.source.voltage.sense.get_source()
```

Sets remote sense detection.

return
arg_0: No help available

set_source(*arg_0: str*) → None

```
# SCPI: [SOURce]:VOLTage:SENSe[:SOURce]
driver.source.voltage.sense.set_source(arg_0 = r1)
```

Sets remote sense detection.

param arg_0

- AUTO: If remote sense detection is set AUTO, the detection and enabling of the voltage sense relay automatically kicks in when the connection of remote sense wires (S+, S-) to the input of the load is applied.
- EXT: If remote sense detection is set EXT, internal voltage sense relay in the instrument is switched on and the connection of remote sense wires (S+, S-) to the input of the load become necessary. Failure to connect remote sense can cause overvoltage or unregulated voltage output from the R&S NGP800.

6.18 Status

class StatusCls

Status commands group definition. 20 total commands, 2 Subgroups, 0 group commands

Subgroups

6.18.1 Operation

class OperationCls

Operation commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Subgroups

6.18.1.1 Instrument

SCPI Commands

```
STATus:OPERation:INSTrument:EVENt
STATus:OPERation:INSTrument:CONDition
STATus:OPERation:INSTrument:PTRansition
STATus:OPERation:INSTrument:NTRansition
STATus:OPERation:INSTrument:ENABle
```

class InstrumentCls

Instrument commands group definition. 10 total commands, 1 Subgroups, 5 group commands

get_condition() → int

```
# SCPI: STATus:OPERation:INSTrument:CONDition
value: int = driver.status.operation.instrument.get_condition()
```

Returns the contents of the CONDition part of the status register to check for operation instrument or measurement states. Reading the CONDition registers does not delete the contents.

return
result: Condition bits in decimal representation.

get_enable() → int

```
# SCPI: STATus:OPERation:INSTrument:ENABle
value: int = driver.status.operation.instrument.get_enable()
```

Controls or queries the ENABle part of the STATus:OPERation register. The ENABle defines which events in the EVENt part of the status register are forwarded to the OPERation summary bit (bit 7) of the status byte. The status byte can be used to create a service request.

return
arg_0: No help available

get_event() → int

```
# SCPI: STATus:OPERation:INSTrument[:EVENt]
value: int = driver.status.operation.instrument.get_event()
```

Returns the contents of the EVENt part of the status register to check whether an event has occurred since the last reading. Reading an EVENt register deletes its contents.

return
result: No help available

get_ntransition() → int

```
# SCPI: STATus:OPERation:INSTrument:NTRansition
value: int = driver.status.operation.instrument.get_ntransition()
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

return
arg_0: No help available

get_ptransition() → int

```
# SCPI: STATus:OPERation:INSTrument:PTRansition
value: int = driver.status.operation.instrument.get_ptransition()
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

return
arg_0: No help available

set_enable(*arg_0: int*) → None

```
# SCPI: STATus:OPERation:INSTrument:ENABle
driver.status.operation.instrument.set_enable(arg_0 = 1)
```

Controls or queries the ENABle part of the STATus:OPERation register. The ENABle defines which events in the EVENt part of the status register are forwarded to the OPERation summary bit (bit 7) of the status byte. The status byte can be used to create a service request.

param arg_0
No help available

set_ntransition(*arg_0: int*) → None

```
# SCPI: STATus:OPERation:INSTrument:NTRansition
driver.status.operation.instrument.set_ntransition(arg_0 = 1)
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

set_ptransition(*arg_0: int*) → None

```
# SCPI: STATus:OPERation:INSTrument:PTRansition
driver.status.operation.instrument.set_ptransition(arg_0 = 1)
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

Subgroups

6.18.1.1.1 Isummary<Channel>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.status.operation.instrument.isummary.repcap_channel_get()
driver.status.operation.instrument.isummary.repcap_channel_set(repcap.Channel.Nr1)
```

class IsummaryCls

Isummary commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

Subgroups

6.18.1.1.1.1 Condition

SCPI Commands

```
STATus:OPERation:INSTrument:ISUMmary<Channel>:CONDition
```

class ConditionCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>:CONDition
value: int = driver.status.operation.instrument.isummary.condition.get(channel
↪= repcap.Channel.Default)
```

Returns the contents of the CONDition part of the status register to check for operation instrument or measurement states. Reading the CONDition registers does not delete the contents.

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

return
result: Condition bits in decimal representation.

6.18.1.1.1.2 Enable

SCPI Commands

```
STATus:OPERation:INSTrument:ISUMmary<Channel>:ENABle
```

class EnableCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>:ENABle
value: int = driver.status.operation.instrument.isummary.enable.get(channel =␣
↪repcap.Channel.Default)
```

Controls or queries the ENABle part of the STATus:OPERation register. The ENABle defines which events in the EVENt part of the status register are forwarded to the OPERation summary bit (bit 7) of the status byte. The status byte can be used to create a service request.

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

return
arg_0: No help available

set(*arg_0: int*, *channel=Channel.Default*) → None

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>:ENABle
driver.status.operation.instrument.isummary.enable.set(arg_0 = 1, channel =␣
↪repcap.Channel.Default)
```

Controls or queries the ENABle part of the STATus:OPERation register. The ENABle defines which events in the EVENt part of the status register are forwarded to the OPERation summary bit (bit 7) of the status byte. The status byte can be used to create a service request.

param arg_0
No help available

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

6.18.1.1.1.3 Event

SCPI Commands

```
STATus:OPERation:INSTrument:ISUMmary<Channel>:EVENt
```

class EventCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>[:EVENt]
value: int = driver.status.operation.instrument.isummary.event.get(channel =␣
↪repcap.Channel.Default)
```

Returns the contents of the EVENt part of the status register to check whether an event has occurred since the last reading. Reading an EVENt register deletes its contents.

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

return
result: No help available

6.18.1.1.1.4 Ntransition

SCPI Commands

```
STATus:OPERation:INSTrument:ISUMmary<Channel>:NTRansition
```

class NtransitionCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>:NTRansition
value: int = driver.status.operation.instrument.isummary.ntransition.
↪get(channel = repcap.Channel.Default)
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

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

return
arg_0: No help available

set(*arg_0: int*, *channel=Channel.Default*) → None

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>:NTRansition
driver.status.operation.instrument.isummary.ntransition.set(arg_0 = 1, channel␣
↪= repcap.Channel.Default)
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

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

6.18.1.1.1.5 Ptransition

SCPI Commands

```
STATus:OPERation:INSTrument:ISUMmary<Channel>:PTRansition
```

class PtransitionCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>:PTRansition
value: int = driver.status.operation.instrument.isummary.ptransition.
↪get(channel = repcap.Channel.Default)
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

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

return
arg_0: No help available

set(*arg_0: int*, *channel=Channel.Default*) → None

```
# SCPI: STATus:OPERation:INSTrument:ISUMmary<Channel>:PTRansition
driver.status.operation.instrument.isummary.ptransition.set(arg_0 = 1, channel␣
↪= repcap.Channel.Default)
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

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

6.18.2 Questionable

class QuestionableCls

Questionable commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Subgroups

6.18.2.1 Instrument

SCPI Commands

```
STATus:QUEStionable:INSTrument:EVENt
STATus:QUEStionable:INSTrument:CONDition
STATus:QUEStionable:INSTrument:PTRansition
STATus:QUEStionable:INSTrument:NTRansition
STATus:QUEStionable:INSTrument:ENABle
```

class InstrumentCls

Instrument commands group definition. 10 total commands, 1 Subgroups, 5 group commands

get_condition() → int

```
# SCPI: STATus:QUEStionable:INSTrument:CONDition
value: int = driver.status.questionable.instrument.get_condition()
```

Returns the contents of the CONDition part of the status register to check for questionable instrument or measurement states. Reading the CONDition registers does not delete the contents.

return
result: Condition bits in decimal representation

get_enable() → int

```
# SCPI: STATus:QUEStionable:INSTrument:ENABle
value: int = driver.status.questionable.instrument.get_enable()
```

Sets or queries the enable mask that allows true conditions in the EVENt part to be reported in the summary bit. If a bit in the ENABle part is 1, and the corresponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reported to the next higher level.

return
arg_0: No help available

get_event() → int

```
# SCPI: STATus:QUEStionable:INSTrument[:EVENt]
value: int = driver.status.questionable.instrument.get_event()
```

Returns the contents of the EVENt part of the status register to check whether an event has occurred since the last reading. Reading an EVENt register deletes its contents.

return
result: Event bits in decimal representation

get_ntransition() → int

```
# SCPI: STATus:QUEStionable:INSTrument:NTRansition
value: int = driver.status.questionable.instrument.get_ntransition()
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

return
arg_0: No help available

get_ptransition() → int

```
# SCPI: STATus:QUEStionable:INSTrument:PTRansition
value: int = driver.status.questionable.instrument.get_ptransition()
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

return
arg_0: No help available

set_enable(*arg_0: int*) → None

```
# SCPI: STATus:QUEStionable:INSTrument:ENABle
driver.status.questionable.instrument.set_enable(arg_0 = 1)
```

Sets or queries the enable mask that allows true conditions in the EVENt part to be reported in the summary bit. If a bit in the ENABle part is 1, and the corresponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reported to the next higher level.

param arg_0
Bit mask in decimal representation

set_ntransition(*arg_0: int*) → None

```
# SCPI: STATus:QUEStionable:INSTrument:NTRansition
driver.status.questionable.instrument.set_ntransition(arg_0 = 1)
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

set_ptransition(*arg_0: int*) → None

```
# SCPI: STATus:QUEStionable:INSTrument:PTRansition
driver.status.questionable.instrument.set_ptransition(arg_0 = 1)
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

Subgroups

6.18.2.1.1 Isummary<Channel>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.status.questionable.instrument.isummary.repcap_channel_get()
driver.status.questionable.instrument.isummary.repcap_channel_set(repcap.Channel.Nr1)
```

class IsummaryCls

Isummary commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

Subgroups

6.18.2.1.1.1 Condition

SCPI Commands

```
STATus:QUEStionable:INSTrument:ISUMmary<Channel>:CONDition
```

class ConditionCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>:CONDition
value: int = driver.status.questionable.instrument.isummary.condition.
↪get(channel = repcap.Channel.Default)
```

Returns the contents of the CONDition part of the status register to check for questionable instrument or measurement states. Reading the CONDition registers does not delete the contents.

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

return
result: Condition bits in decimal representation

6.18.2.1.1.2 Enable

SCPI Commands

```
STATus:QUEStionable:INSTrument:ISUMmary<Channel>:ENABle
```

class EnableCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>:ENABle
value: int = driver.status.questionable.instrument.isummary.enable.get(channel␣
↪= repcap.Channel.Default)
```

Sets or queries the enable mask that allows true conditions in the EVENt part to be reported in the summary bit. If a bit in the ENABle part is 1, and the corresponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reported to the next higher level.

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

return
arg_0: No help available

set(*arg_0: int*, *channel=Channel.Default*) → None

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>:ENABle
driver.status.questionable.instrument.isummary.enable.set(arg_0 = 1, channel =␣
↪repcap.Channel.Default)
```

Sets or queries the enable mask that allows true conditions in the EVENt part to be reported in the summary bit. If a bit in the ENABle part is 1, and the corresponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reported to the next higher level.

param arg_0
Bit mask in decimal representation

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

6.18.2.1.1.3 Event

SCPI Commands

```
STATus:QUEStionable:INSTrument:ISUMmary<Channel>:EVENt
```

class EventCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>[:EVENt]
value: int = driver.status.questionable.instrument.isummary.event.get(channel =␣
↪repcap.Channel.Default)
```

Returns the contents of the EVENt part of the status register to check whether an event has occurred since the last reading. Reading an EVENt register deletes its contents.

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

return
result: Event bits in decimal representation

6.18.2.1.1.4 Ntransition

SCPI Commands

```
STATus:QUEStionable:INSTrument:ISUMmary<Channel>:NTRansition
```

class NtransitionCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>:NTRansition
value: int = driver.status.questionable.instrument.isummary.ntransition.
↪get(channel = repcap.Channel.Default)
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

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

return
arg_0: No help available

set(*arg_0: int*, *channel=Channel.Default*) → None

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>:NTRansition
driver.status.questionable.instrument.isummary.ntransition.set(arg_0 = 1,␣
↪channel = repcap.Channel.Default)
```

Sets or queries the negative transition filter. Setting a bit in the negative transition filter shall cause a 1 to 0 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

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

6.18.2.1.1.5 Ptransition

SCPI Commands

```
STATus:QUEStionable:INSTrument:ISUMmary<Channel>:PTRansition
```

class PtransitionCls

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

get(*channel=Channel.Default*) → int

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>:PTRansition
value: int = driver.status.questionable.instrument.isummary.ptransition.
↪get(channel = repcap.Channel.Default)
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

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

return
arg_0: No help available

set(*arg_0: int*, *channel=Channel.Default*) → None

```
# SCPI: STATus:QUEStionable:INSTrument:ISUMmary<Channel>:PTRansition
driver.status.questionable.instrument.isummary.ptransition.set(arg_0 = 1,␣
↪channel = repcap.Channel.Default)
```

Sets or queries the positive transition filter. Setting a bit in the positive transition filter shall cause a 0 to 1 transition in the corresponding bit of the associated condition register to cause a 1 to be written in the associated bit of the corresponding event register.

param arg_0
No help available

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

6.19 System

SCPI Commands

```
SYSTem:UPTime
```

class SystemCls

System commands group definition. 46 total commands, 12 Subgroups, 1 group commands

get_up_time() → str

```
# SCPI: SYSTem:UPTime
value: str = driver.system.get_up_time()
```

Queries system uptime.

return
result: No help available

Subgroups

6.19.1 Beeper

class BeeperCls

Beeper commands group definition. 8 total commands, 5 Subgroups, 0 group commands

Subgroups

6.19.1.1 Complete

SCPI Commands

```
SYSTem:BEEPer:COMPlete:STATe
```

class CompleteCls

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

get_state() → bool

```
# SCPI: SYSTem:BEEPer[:COMPlete]:STATe
value: bool = driver.system.beeper.complete.get_state()
```

Sets or queries the beeper tone.

return
beeper_state: No help available

set_state(*beeper_state: bool*) → None

```
# SCPI: SYSTem:BEEPer[:COMPlete]:STATe
driver.system.beeper.complete.set_state(beeper_state = False)
```

Sets or queries the beeper tone.

param beeper_state
1 | 0 ON OFF - Control beeper is deactivated. OFF ON - Control beeper is activated.

Subgroups

6.19.1.1.1 Immediate

SCPI Commands

```
SYSTem:BEEPer:COMPlete:IMMediate
```

class ImmediateCls

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

set() → None

```
# SCPI: SYSTem:BEEPer[:COMPlete][:IMMediate]
driver.system.beeper.complete.immediate.set()
```

No command help available

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

```
# SCPI: SYSTem:BEEPer[:COMPlete][:IMMediate]
driver.system.beeper.complete.immediate.set_with_opc()
```

No command help available

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

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

6.19.1.2 Current

SCPI Commands

```
SYSTem:BEEPer:CURRent:STATe
```

class CurrentCls

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

get_state() → bool

```
# SCPI: SYSTem:BEEPer:CURRent:STATe
value: bool = driver.system.beeper.current.get_state()
```

Sets or queries current control beeper tone state.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: SYSTem:BEEPer:CURRent:STATe
driver.system.beeper.current.set_state(arg_0 = False)
```

Sets or queries current control beeper tone state.

param arg_0

- 1: Control beeper is activated.
- 0: Control beeper is deactivated.

6.19.1.3 Output

SCPI Commands

```
SYSTem:BEEPer:OUTPut:STATe
```

class OutputCls

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

get_state() → bool

```
# SCPI: SYSTem:BEEPer:OUTPut:STATe
value: bool = driver.system.beeper.output.get_state()
```

Sets or queries output beeper tone state.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: SYSTem:BEEPer:OUTPut:STATe
driver.system.beeper.output.set_state(arg_0 = False)
```

Sets or queries output beeper tone state.

param arg_0

- 1: Output beeper is activated.
- 0: Output beeper is deactivated.

6.19.1.4 Protection

SCPI Commands

```
SYSTem:BEEPer:PROTection:STATe
```

class ProtectionCls

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

get_state() → bool

```
# SCPI: SYSTem:BEEPer:PROTection:STATe
value: bool = driver.system.beeper.protection.get_state()
```

Sets or queries protection beeper tone state.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: SYSTem:BEEPer:PROTection:STATe
driver.system.beeper.protection.set_state(arg_0 = False)
```

Sets or queries protection beeper tone state.

param arg_0

- 1: Protection beeper is activated.
- 0: Protection beeper is deactivated.

Subgroups

6.19.1.4.1 Immediate

SCPI Commands

```
SYSTem:BEEPer:PROTection:IMMediate
```

class ImmediateCls

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

set() → None

```
# SCPI: SYSTem:BEEPer:PROTection[:IMMediate]
driver.system.beeper.protection.immediate.set()
```

Returns a single protection beep immediately.

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

```
# SCPI: SYSTem:BEEPer:PROTection[:IMMediate]
driver.system.beeper.protection.immediate.set_with_opc()
```

Returns a single protection beep immediately.

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

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

6.19.1.5 WarningPy

SCPI Commands

```
SYSTem:BEEPer:WARNing:STATe
```

class WarningPyCls

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

get_state() → bool

```
# SCPI: SYSTem:BEEPer:WARNing:STATe
value: bool = driver.system.beeper.warningPy.get_state()
```

Sets or queries ‘error/warning’ beeper tone state.

return
warning_beep_state: No help available

set_state(*warning_beep_state: bool*) → None

```
# SCPI: SYSTem:BEEPer:WARNing:STATe
driver.system.beeper.warningPy.set_state(warning_beep_state = False)
```

Sets or queries ‘error/warning’ beeper tone state.

param warning_beep_state

- 1: Beep sound for ‘error/warning’ is enabled.
- 0: Beep sound for ‘error/warning’ is disabled.

Subgroups

6.19.1.5.1 Immediate

SCPI Commands

```
SYSTem:BEEPer:WARNing:IMMediate
```

class ImmediateCls

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

set() → None

```
# SCPI: SYSTem:BEEPer:WARNing[:IMMediate]
driver.system.beeper.warningPy.immediate.set()
```

Returns a single ‘error/warning’ beep immediately.

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

```
# SCPI: SYSTem:BEEPer:WARNing[:IMMediate]
driver.system.beeper.warningPy.immediate.set_with_opc()
```

Returns a single 'error/warning' beep immediately.

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

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

6.19.2 Communicate

class CommunicateCls

Communicate commands group definition. 26 total commands, 3 Subgroups, 0 group commands

Subgroups

6.19.2.1 Lan

SCPI Commands

```
SYSTem:COMMunicate:LAN:DHCP
SYSTem:COMMunicate:LAN:ADDRess
SYSTem:COMMunicate:LAN:SMASk
SYSTem:COMMunicate:LAN:DGATeway
SYSTem:COMMunicate:LAN:HOSTname
SYSTem:COMMunicate:LAN:MAC
SYSTem:COMMunicate:LAN:RESet
SYSTem:COMMunicate:LAN:EDITed
```

class LanCls

Lan commands group definition. 10 total commands, 2 Subgroups, 8 group commands

get_address() → str

```
# SCPI: SYSTem:COMMunicate:LAN:ADDRess
value: str = driver.system.communicate.lan.get_address()
```

No command help available

return
arg_0: No help available

get_dgateway() → str

```
# SCPI: SYSTem:COMMunicate:LAN:DGATeway
value: str = driver.system.communicate.lan.get_dgateway()
```

No command help available

return
arg_0: No help available

get_dhcp() → bool

```
# SCPI: SYSTem:COMMunicate:LAN:DHCP
value: bool = driver.system.communicate.lan.get_dhcp()
```

No command help available

return
arg_0: No help available

get_edited() → str

```
# SCPI: SYSTem:COMMunicate:LAN:EDITed
value: str = driver.system.communicate.lan.get_edited()
```

No command help available

return
result: No help available

get_hostname() → str

```
# SCPI: SYSTem:COMMunicate:LAN:HOSTname
value: str = driver.system.communicate.lan.get_hostname()
```

No command help available

return
arg_0: No help available

get_mac() → str

```
# SCPI: SYSTem:COMMunicate:LAN:MAC
value: str = driver.system.communicate.lan.get_mac()
```

No command help available

return
result: No help available

get_smask() → str

```
# SCPI: SYSTem:COMMunicate:LAN:SMASk
value: str = driver.system.communicate.lan.get_smask()
```

No command help available

return
arg_0: No help available

reset() → None

```
# SCPI: SYSTem:COMMunicate:LAN:RESet
driver.system.communicate.lan.reset()
```

No command help available

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

```
# SCPI: SYSTem:COMMunicate:LAN:RESet
driver.system.communicate.lan.reset_with_opc()
```

No command help available

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

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

set_address(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:LAN:ADDRess
driver.system.communicate.lan.set_address(arg_0 = '1')
```

No command help available

param arg_0
No help available

set_dgateway(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:LAN:DGATeway
driver.system.communicate.lan.set_dgateway(arg_0 = '1')
```

No command help available

param arg_0
No help available

set_dhcp(*arg_0: bool*) → None

```
# SCPI: SYSTem:COMMunicate:LAN:DHCP
driver.system.communicate.lan.set_dhcp(arg_0 = False)
```

No command help available

param arg_0
No help available

set_hostname(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:LAN:HOSTname
driver.system.communicate.lan.set_hostname(arg_0 = '1')
```

No command help available

param arg_0
No help available

set_smask(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:LAN:SMASk
driver.system.communicate.lan.set_smask(arg_0 = '1')
```

No command help available

param arg_0
No help available

Subgroups

6.19.2.1.1 Apply

SCPI Commands

```
SYSTem:COMMunicate:LAN:APPLy
```

class ApplyCls

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

set() → None

```
# SCPI: SYSTem:COMMunicate:LAN:APPLy
driver.system.communicate.lan.apply.set()
```

No command help available

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

```
# SCPI: SYSTem:COMMunicate:LAN:APPLy
driver.system.communicate.lan.apply.set_with_opc()
```

No command help available

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

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

6.19.2.1.2 Discard

SCPI Commands

```
SYSTem:COMMunicate:LAN:DISCard
```

class DiscardCls

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

set() → None

```
# SCPI: SYSTem:COMMunicate:LAN:DISCard
driver.system.communicate.lan.discard.set()
```

No command help available

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

```
# SCPI: SYSTem:COMMunicate:LAN:DISCard
driver.system.communicate.lan.discard.set_with_opc()
```

No command help available

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

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

6.19.2.2 Socket

SCPI Commands

```
SYSTem:COMMunicate:SOCKet:DHCP
SYSTem:COMMunicate:SOCKet:IPADdress
SYSTem:COMMunicate:SOCKet:MASK
SYSTem:COMMunicate:SOCKet:GATeway
SYSTem:COMMunicate:SOCKet:RESet
```

class SocketCls

Socket commands group definition. 7 total commands, 2 Subgroups, 5 group commands

get_dhcp() → bool

```
# SCPI: SYSTem:COMMunicate:SOCKet:DHCP
value: bool = driver.system.communicate.socket.get_dhcp()
```

Sets the LAN interface mode.

return
arg_0: No help available

get_gateway() → str

```
# SCPI: SYSTem:COMMunicate:SOCKet:GATeway
value: str = driver.system.communicate.socket.get_gateway()
```

Sets or queries gateway for LAN.

return
arg_0: No help available

get_ip_address() → str

```
# SCPI: SYSTem:COMMunicate:SOCKet:IPADdress
value: str = driver.system.communicate.socket.get_ip_address()
```

Sets or queries IP address of the LAN interface.

return
arg_0: No help available

get_mask() → str

```
# SCPI: SYSTem:COMMunicate:SOCKet:MASK
value: str = driver.system.communicate.socket.get_mask()
```

Sets or queries the subnet mask for LAN.

return
arg_0: No help available

reset() → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:RESet
driver.system.communicate.socket.reset()
```

Resets LAN settings.

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

```
# SCPI: SYSTem:COMMunicate:SOCKet:RESet
driver.system.communicate.socket.reset_with_opc()
```

Resets LAN settings.

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

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

set_dhcp(*arg_0: bool*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:DHCP
driver.system.communicate.socket.set_dhcp(arg_0 = False)
```

Sets the LAN interface mode.

param arg_0

- 1: DHCP is enabled.Automatic IP address from DHCP server.
- 0: DHCP is disabled.Manually set IP address.

set_gateway(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:GATeway
driver.system.communicate.socket.set_gateway(arg_0 = '1')
```

Sets or queries gateway for LAN.

param arg_0
Gateway address.

set_ip_address(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:IPADdress
driver.system.communicate.socket.set_ip_address(arg_0 = '1')
```

Sets or queries IP address of the LAN interface.

param arg_0
IP address.

set_mask(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:MASK
driver.system.communicate.socket.set_mask(arg_0 = '1')
```

Sets or queries the subnet mask for LAN.

param arg_0
Subnet address.

Subgroups

6.19.2.2.1 Apply

SCPI Commands

```
SYSTem:COMMunicate:SOCKet:APPLy
```

class ApplyCls

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

set() → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:APPLy
driver.system.communicate.socket.apply.set()
```

Apply LAN configuration settings.

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

```
# SCPI: SYSTem:COMMunicate:SOCKet:APPLy
driver.system.communicate.socket.apply.set_with_opc()
```

Apply LAN configuration settings.

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

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

6.19.2.2.2 Discard

SCPI Commands

```
SYSTem:COMMunicate:SOCKet:DISCard
```

class DiscardCls

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

set() → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:DISCard
driver.system.communicate.socket.discard.set()
```

Discards LAN settings.

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

```
# SCPI: SYSTem:COMMunicate:SOCKet:DISCard
driver.system.communicate.socket.discard.set_with_opc()
```

Discards LAN settings.

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

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

6.19.2.3 Wlan

SCPI Commands

```
SYSTem:COMMunicate:WLAN:STATe
SYSTem:COMMunicate:WLAN:SSID
SYSTem:COMMunicate:WLAN:PASSword
SYSTem:COMMunicate:WLAN:DHCP
SYSTem:COMMunicate:WLAN:IPADdress
SYSTem:COMMunicate:WLAN:MASK
SYSTem:COMMunicate:WLAN:GATeway
```

class WlanCls

Wlan commands group definition. 9 total commands, 2 Subgroups, 7 group commands

get_dhcp() → bool

```
# SCPI: SYSTem:COMMunicate:WLAN:DHCP
value: bool = driver.system.communicate.wlan.get_dhcp()
```

No command help available

return
arg_0: No help available

get_gateway() → str

```
# SCPI: SYSTem:COMMunicate:WLAN:GATeway
value: str = driver.system.communicate.wlan.get_gateway()
```

No command help available

return
arg_0: No help available

get_ip_address() → str

```
# SCPI: SYSTem:COMMunicate:WLAN:IPADdress
value: str = driver.system.communicate.wlan.get_ip_address()
```

Queries IP address for WLAN. Available only if option R&S NGP-K102.

return
arg_0: No help available

get_mask() → str

```
# SCPI: SYSTem:COMMunicate:WLAN:MASK
value: str = driver.system.communicate.wlan.get_mask()
```

No command help available

return
arg_0: No help available

get_ssid() → str

```
# SCPI: SYSTem:COMMunicate:WLAN:SSID
value: str = driver.system.communicate.wlan.get_ssid()
```

Sets or queries SSID of the access point when wireless interface works as a client. Available only if option R&S NGP-K102.

return
arg_0: No help available

get_state() → bool

```
# SCPI: SYSTem:COMMunicate:WLAN[:STATe]
value: bool = driver.system.communicate.wlan.get_state()
```

Enables or disables WLAN state. Available only if option R&S NGP-K102.

return
arg_0: No help available

set_dhcp(*arg_0: bool*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN:DHCP
driver.system.communicate.wlan.set_dhcp(arg_0 = False)
```

No command help available

param arg_0
No help available

set_gateway(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN:GATeway
driver.system.communicate.wlan.set_gateway(arg_0 = '1')
```

No command help available

param arg_0
No help available

set_ip_address(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN:IPADdress
driver.system.communicate.wlan.set_ip_address(arg_0 = '1')
```

Queries IP address for WLAN. Available only if option R&S NGP-K102.

param arg_0
No help available

set_mask(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN:MASK
driver.system.communicate.wlan.set_mask(arg_0 = '1')
```

No command help available

param arg_0
No help available

set_password(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN:PASSword
driver.system.communicate.wlan.set_password(arg_0 = '1')
```

Sets or queries password for WLAN. Available only if option R&S NGP-K102.

param arg_0
WLAN password.

set_ssid(*arg_0: str*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN:SSID
driver.system.communicate.wlan.set_ssid(arg_0 = '1')
```

Sets or queries SSID of the access point when wireless interface works as a client. Available only if option R&S NGP-K102.

param arg_0
SSID of access point.

set_state(*arg_0: bool*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN[:STATe]
driver.system.communicate.wlan.set_state(arg_0 = False)
```

Enables or disables WLAN state. Available only if option R&S NGP-K102.

param arg_0

- 1: Enable WLAN.
- 0: Disable WLAN.

Subgroups

6.19.2.3.1 Apply

SCPI Commands

```
SYSTem:COMMunicate:WLAN:APPLy
```

class ApplyCls

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

set() → None

```
# SCPI: SYSTem:COMMunicate:WLAN:APPLy
driver.system.communicate.wlan.apply.set()
```

No command help available

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

```
# SCPI: SYSTem:COMMunicate:WLAN:APPLy
driver.system.communicate.wlan.apply.set_with_opc()
```

No command help available

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

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

6.19.2.3.2 Connection

SCPI Commands

```
SYSTem:COMMunicate:WLAN:CONNection:STATe
```

class ConnectionCls

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

get_state() → bool

```
# SCPI: SYSTem:COMMunicate:WLAN:CONNection[:STATe]
value: bool = driver.system.communicate.wlan.connection.get_state()
```

Connects or disconnects WLAN to the predefined wireless access point. Available only if option R&S NGP-K102.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: SYSTem:COMMunicate:WLAN:CONNection[:STATe]
driver.system.communicate.wlan.connection.set_state(arg_0 = False)
```

Connects or disconnects WLAN to the predefined wireless access point. Available only if option R&S NGP-K102.

param arg_0

- 1: Connect WLAN to the predefined wireless access point.
- 0: Disconnect WLAN from the predefined wireless access point.

6.19.3 Date

SCPI Commands

```
SYSTem:DATE
```

class DateCls

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

class DateStruct

Response structure. Fields:

- Year: float: Sets year of the date.
- Month: float: Sets month of the date.
- Day: float: No parameter help available

get() → DateStruct

```
# SCPI: SYSTem:DATE
value: DateStruct = driver.system.date.get()
```

Sets or queries the system date.

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

set(*year: float*, *month: float*, *day: float*) → None

```
# SCPI: SYSTem:DATE
driver.system.date.set(year = 1.0, month = 1.0, day = 1.0)
```

Sets or queries the system date.

param year
Sets year of the date.

param month
Sets month of the date.

param day
Sets day of the date.

6.19.4 Key

SCPI Commands

```
SYSTem:KEY:BRIGhtness
```

class KeyCls

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

get_brightness() → float

```
# SCPI: SYSTem:KEY:BRIGhtness
value: float = driver.system.key.get_brightness()
```

Sets or queries the front panel key brightness.

return
front_key_brightness: No help available

set_brightness(*front_key_brightness: float*) → None

```
# SCPI: SYSTem:KEY:BRIGhtness
driver.system.key.set_brightness(front_key_brightness = 1.0)
```

Sets or queries the front panel key brightness.

param front_key_brightness
Sets the key brightness.

6.19.5 Local

SCPI Commands

```
SYSTem:LOCal
```

class LocalCls

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

set() → None

```
# SCPI: SYSTem:LOCal
driver.system.local.set()
```

Sets the system to front panel control. The front panel control is unlocked.

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

```
# SCPI: SYSTem:LOCal
driver.system.local.set_with_opc()
```

Sets the system to front panel control. The front panel control is unlocked.

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

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

6.19.6 Remote

SCPI Commands

```
SYSTem:REMote
```

class RemoteCls

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

set() → None

```
# SCPI: SYSTem:REMote
driver.system.remote.set()
```

Sets the system to remote state. The front panel control is locked.

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

```
# SCPI: SYSTem:REMote
driver.system.remote.set_with_opc()
```

Sets the system to remote state. The front panel control is locked.

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

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

6.19.7 Restart

SCPI Commands

```
SYSTem:RESTart
```

class RestartCls

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

set() → None

```
# SCPI: SYSTem:RESTart
driver.system.restart.set()
```

No command help available

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

```
# SCPI: SYSTem:RESTart
driver.system.restart.set_with_opc()
```

No command help available

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

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

6.19.8 RwLock

SCPI Commands

```
SYSTem:RWLock
```

class RwLockCls

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

set() → None

```
# SCPI: SYSTem:RWLock
driver.system.rwLock.set()
```

Sets the system to remote state. The front panel control is locked. You are only able to unlock the front panel control via SCPI command SYSTem:LOCal.

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

```
# SCPI: SYSTem:RWLock
driver.system.rwLock.set_with_opc()
```

Sets the system to remote state. The front panel control is locked. You are only able to unlock the front panel control via SCPI command SYSTem:LOCal.

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

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

6.19.9 Setting

class SettingCls

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

Subgroups

6.19.9.1 Default

SCPI Commands

```
SYSTem:SETTing:DEFault:SAVE
```

class DefaultCls

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

save(*file_path: Optional[str] = None*) → None

```
# SCPI: SYSTem:SETTing:DEFault:SAVE
driver.system.setting.default.save(file_path = '1')
```

No command help available

param file_path
No help available

6.19.10 Time

SCPI Commands

```
SYSTem:TIME
```

class TimeCls

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

class TimeStruct

Response structure. Fields:

- Hour: int: No parameter help available
- Minute: int: No parameter help available
- Second: int: No parameter help available

get() → TimeStruct

```
# SCPI: SYSTem:TIME
value: TimeStruct = driver.system.time.get()
```

Sets or queries the system time.

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

set(*hour: int*, *minute: int*, *second: int*) → None

```
# SCPI: SYSTem:TIME
driver.system.time.set(hour = 1, minute = 1, second = 1)
```

Sets or queries the system time.

param hour
No help available

param minute
No help available

param second
No help available

6.19.11 Touch

SCPI Commands

```
SYSTem:TOUCh:STATe
```

class TouchCls

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

get_state() → bool

```
# SCPI: SYSTem:TOUCh[:STATe]
value: bool = driver.system.touch.get_state()
```

Enables or disables touch interface beep.

return
arg_0: No help available

set_state(*arg_0: bool*) → None

```
# SCPI: SYSTem:TOUCh[:STATe]
driver.system.touch.set_state(arg_0 = False)
```

Enables or disables touch interface beep.

param arg_0

- 1: Touch interface beep is activated.
- 0: Touch interface beep is deactivated.

6.19.12 Vnc

SCPI Commands

```
SYSTem:VNC:STATe
SYSTem:VNC:PORT
```

class VncCls

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

get_port() → int

```
# SCPI: SYSTem:VNC:PORT
value: int = driver.system.vnc.get_port()
```

Sets or queries the VNC port number.

return
arg_0: No help available

get_state() → bool

```
# SCPI: SYSTem:VNC:STATe
value: bool = driver.system.vnc.get_state()
```

Enables or disables VNC state.

return
arg_0: No help available

set_port(*arg_0: int*) → None

```
# SCPI: SYSTem:VNC:PORT
driver.system.vnc.set_port(arg_0 = 1)
```

Sets or queries the VNC port number.

param arg_0
No help available

set_state(*arg_0: bool*) → None

```
# SCPI: SYSTem:VNC:STATe
driver.system.vnc.set_state(arg_0 = False)
```

Enables or disables VNC state.

param arg_0

- 1: Enable VNC.
- 0: Disable VNC.

6.20 Tracking

class TrackingCls

Tracking commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Subgroups

6.20.1 Enable

SCPI Commands

```
TRACking:ENABle:GENeral
```

class EnableCls

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

get_general() → bool

```
# SCPI: TRACking[:ENABle]:GENeral
value: bool = driver.tracking.enable.get_general()
```

Sets or queries the status of the master tracking state.

return
arg_0: - 0: Master tracking is disabled - 1: Master tracking is enabled

set_general(*arg_0: bool*) → None

```
# SCPI: TRACking[:ENABle]:GENeral
driver.tracking.enable.set_general(arg_0 = False)
```

Sets or queries the status of the master tracking state.

param arg_0

- 0: Master tracking is disabled
- 1: Master tracking is enabled

Subgroups

6.20.1.1 Ch

SCPI Commands

```
TRACking:ENABle:CH<Channel>
```

class ChCls

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

get(*channel=Channel.Nr1*) → bool

```
# SCPI: TRACking[:ENABle]:CH<CHANNEL>
value: bool = driver.tracking.enable.ch.get(channel = repcap.Channel.Nr1)
```

Sets or queries the tracking status on selected channel.

param channel
optional repeated capability selector. Default value: Nr1

return
arg_0: - 0: Tracking is disabled on specified channel. - 1: Tracking is enabled on specified channel.

set(*arg_0: bool*, *channel=Channel.Nr1*) → None

```
# SCPI: TRACking[:ENABle]:CH<CHANNEL>
driver.tracking.enable.ch.set(arg_0 = False, channel = repcap.Channel.Nr1)
```

Sets or queries the tracking status on selected channel.

param arg_0

- 0: Tracking is disabled on specified channel.
- 1: Tracking is enabled on specified channel.

param channel
optional repeated capability selector. Default value: Nr1

6.20.1.2 Select

class SelectCls

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

Subgroups

6.20.1.2.1 Ch

SCPI Commands

```
TRACking:ENABle:SELect:CH<Channel>
```

class ChCls

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

get(*channel=Channel.Nr1*) → bool

```
# SCPI: TRACking[:ENABle]:SELect:CH<CHANNEL>
value: bool = driver.tracking.enable.select.ch.get(channel = repcap.Channel.Nr1)
```

Sets or queries the status of tracking soft enable on specific channel.

param channel
optional repeated capability selector. Default value: Nr1

return
arg_0: - 0: Tracking is disabled - 1: Tracking is enabled

set(*arg_0: bool*, *channel=Channel.Nr1*) → None

```
# SCPI: TRACking[:ENABle]:SELect:CH<CHANNEL>
driver.tracking.enable.select.ch.set(arg_0 = False, channel = repcap.Channel.
↪Nr1)
```

Sets or queries the status of tracking soft enable on specific channel.

param arg_0

- 0: Tracking is disabled
- 1: Tracking is enabled

param channel
optional repeated capability selector. Default value: Nr1

6.21 Trigger

class TriggerCls

Trigger commands group definition. 14 total commands, 7 Subgroups, 0 group commands

Subgroups

6.21.1 Channel

class ChannelCls

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

Subgroups

6.21.1.1 Dio<DigitalIo>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.trigger.channel.dio.repcap_digitalIo_get()
driver.trigger.channel.dio.repcap_digitalIo_set(repcap.DigitalIo.Nr1)
```

SCPI Commands

```
TRIGger:CHANnel:DIO<DigitalIo>
```

class DioCls

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

get(*digitalIo=DigitalIo.Default*) → TriggerChannel

```
# SCPI: TRIGger:CHANnel:DIO<IO>
value: enums.TriggerChannel = driver.trigger.channel.dio.get(digitalIo = repcap.
↪DigitalIo.Default)
```

No command help available

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

return
arg_0: No help available

set(*arg_0: TriggerChannel*, *digitalIo=DigitalIo.Default*) → None

```
# SCPI: TRIGger:CHANnel:DIO<IO>
driver.trigger.channel.dio.set(arg_0 = enums.TriggerChannel.ALL, digitalIo =␣
↪repcap.DigitalIo.Default)
```

No command help available

param arg_0

- NONE: No channel is set as the trigger channel.
- CH1: Ch 1 is set as the trigger channel.
- CH2: Ch 2 is set as the trigger channel.
- CH3: Ch 3 is set as the trigger channel.
- CH4: Ch 4 is set as the trigger channel.
- CHALI: All channels are set as the trigger channel.

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

6.21.2 Condition

class ConditionCls

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

Subgroups

6.21.2.1 Dio<DigitalIo>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.trigger.condition.dio.repcap_digitalIo_get()
driver.trigger.condition.dio.repcap_digitalIo_set(repcap.DigitalIo.Nr1)
```

SCPI Commands

```
TRIGger:CONDition:DIO<DigitalIo>
```

class DioCls

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

class DioStruct

Response structure. Fields:

- Arg_0: enums.TriggerCondition: No parameter help available
- Arg_1: float: No parameter help available

get(*digitalIo=DigitalIo.Default*) → DioStruct

```
# SCPI: TRIGger:CONDition:DIO<IO>
value: DioStruct = driver.trigger.condition.dio.get(digitalIo = repcap.
↪DigitalIo.Default)
```

Sets the trigger condition of the specified Digital I/O line.

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

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

set(*arg_0: TriggerCondition*, *arg_1: Optional[float] = None*, *digitalIo=DigitalIo.Default*) → None

```
# SCPI: TRIGger:CONDition:DIO<IO>
driver.trigger.condition.dio.set(arg_0 = enums.TriggerCondition.ANINput, arg_1
↪= 1.0, digitalIo = repcap.DigitalIo.Default)
```

Sets the trigger condition of the specified Digital I/O line.

param arg_0
No help available

param arg_1
No help available

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

6.21.3 Direction

class DirectionCls

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

Subgroups

6.21.3.1 Dio<DigitalIo>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.trigger.direction.dio.repcap_digitalIo_get()
driver.trigger.direction.dio.repcap_digitalIo_set(repcap.DigitalIo.Nr1)
```

SCPI Commands

```
TRIGger:DIRection:DIO<DigitalIo>
```

class DioCls

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

get(*digitalIo=DigitalIo.Default*) → TriggerDirection

```
# SCPI: TRIGger:DIRection:DIO<IO>
value: enums.TriggerDirection = driver.trigger.direction.dio.get(digitalIo =␣
↪repcap.DigitalIo.Default)
```

Sets or queries the specified Digital I/O line to function as Trigger Input/Output.

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

return
arg_0: No help available

set(*arg_0: TriggerDirection*, *digitalIo=DigitalIo.Default*) → None

```
# SCPI: TRIGger:DIRection:DIO<IO>
driver.trigger.direction.dio.set(arg_0 = enums.TriggerDirection.INPut,␣
↪digitalIo = repcap.DigitalIo.Default)
```

Sets or queries the specified Digital I/O line to function as Trigger Input/Output.

param arg_0
No help available

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

6.21.4 Enable

SCPI Commands

```
TRIGger:ENABle:GENeral
```

class EnableCls

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

get_general() → bool

```
# SCPI: TRIGger[:ENABle]:GENeral
value: bool = driver.trigger.enable.get_general()
```

Sets or queries the enable state of the master on/off of Digital I/O trigger.

return
arg_0: No help available

set_general(*arg_0: bool*) → None

```
# SCPI: TRIGger[:ENABle]:GENeral
driver.trigger.enable.set_general(arg_0 = False)
```

Sets or queries the enable state of the master on/off of Digital I/O trigger.

param arg_0

- 1: Master state of Digital I/O trigger is enabled.

- 0: Master state of Digital I/O trigger is disabled.

Subgroups

6.21.4.1 Dio<DigitalIo>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.trigger.enable.dio.repcap_digitalIo_get()
driver.trigger.enable.dio.repcap_digitalIo_set(repcap.DigitalIo.Nr1)
```

SCPI Commands

```
TRIGger:ENABle:DIO<DigitalIo>
```

class DioCls

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

get(*digitalIo=DigitalIo.Default*) → bool

```
# SCPI: TRIGger[:ENABle]:DIO<IO>
value: bool = driver.trigger.enable.dio.get(digitalIo = repcap.DigitalIo.
↪Default)
```

Sets or queries the enable state of the specified Digital I/O line.

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

return
arg_0: No help available

set(*arg_0: bool*, *digitalIo=DigitalIo.Default*) → None

```
# SCPI: TRIGger[:ENABle]:DIO<IO>
driver.trigger.enable.dio.set(arg_0 = False, digitalIo = repcap.DigitalIo.
↪Default)
```

Sets or queries the enable state of the specified Digital I/O line.

param arg_0

- 1: Selected Digital /O line is enabled.
- 0: Selected Digital /O line is disabled.

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

6.21.4.2 Select

class SelectCls

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

Subgroups

6.21.4.2.1 Dio<DigitalIo>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.trigger.enable.select.dio.repcap_digitalIo_get()
driver.trigger.enable.select.dio.repcap_digitalIo_set(repcap.DigitalIo.Nr1)
```

SCPI Commands

```
TRIGger:ENABle:SELect:DIO<DigitalIo>
```

class DioCls

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

get(*digitalIo=DigitalIo.Default*) → bool

```
# SCPI: TRIGger[:ENABle]:SELect:DIO<IO>
value: bool = driver.trigger.enable.select.dio.get(digitalIo = repcap.DigitalIo.
↪Default)
```

Sets or queries the enable state of the specified Digital I/O line.

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

return
arg_0: No help available

set(*arg_0: bool*, *digitalIo=DigitalIo.Default*) → None

```
# SCPI: TRIGger[:ENABle]:SELect:DIO<IO>
driver.trigger.enable.select.dio.set(arg_0 = False, digitalIo = repcap.
↪DigitalIo.Default)
```

Sets or queries the enable state of the specified Digital I/O line.

param arg_0

- 1: The specified Digital I/O line is enabled.
- 0: The specified Digital I/O line is disabled.

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

6.21.5 Logic

class LogicCls

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

Subgroups

6.21.5.1 Dio<DigitalIo>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.trigger.logic.dio.repcap_digitalIo_get()
driver.trigger.logic.dio.repcap_digitalIo_set(repcap.DigitalIo.Nr1)
```

SCPI Commands

```
TRIGger:LOGic:DIO<DigitalIo>
```

class DioCls

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

get(*digitalIo=DigitalIo.Default*) → LowHigh

```
# SCPI: TRIGger:LOGic:DIO<IO>
value: enums.LowHigh = driver.trigger.logic.dio.get(digitalIo = repcap.
↪DigitalIo.Default)
```

Sets or queries the trigger logic (Active High/Active Low) of the specified Digital I/O line.

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

return
arg_0: No help available

set(*arg_0: LowHigh*, *digitalIo=DigitalIo.Default*) → None

```
# SCPI: TRIGger:LOGic:DIO<IO>
driver.trigger.logic.dio.set(arg_0 = enums.LowHigh.HIGH, digitalIo = repcap.
↪DigitalIo.Default)
```

Sets or queries the trigger logic (Active High/Active Low) of the specified Digital I/O line.

param arg_0
No help available

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

6.21.6 Sequence

class SequenceCls

Sequence commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Subgroups

6.21.6.1 Immediate

class ImmediateCls

Immediate commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Subgroups

6.21.6.1.1 Source

SCPI Commands

```
TRIGger:SEQuence:IMMediate:SOURce
```

class SourceCls

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

get(*arg_0: TriggerSource*) → TriggerSource

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce
value: enums.TriggerSource = driver.trigger.sequence.immediate.source.get(arg_0␣
↪= enums.TriggerSource.DIO)
```

Sets or queries the trigger source. See Figure 'Overview of trigger IO system'.

param arg_0
OUTPut | OMODe | DIO OUTPut Trigger source is from the output channel. OMODe Trigger source is from the different modes (CC, CR, CV, Sink, OVP, OCP, OPP and OTP) detected from the output channel. DIO Trigger source is from DIO connector at the instrument rear panel.

return
arg_0: OUTPut | OMODe | DIO OUTPut Trigger source is from the output channel. OMODe Trigger source is from the different modes (CC, CR, CV, Sink, OVP, OCP, OPP and OTP) detected from the output channel. DIO Trigger source is from DIO connector at the instrument rear panel.

set(*arg_0: TriggerSource*) → None

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce
driver.trigger.sequence.immediate.source.set(arg_0 = enums.TriggerSource.DIO)
```

Sets or queries the trigger source. See Figure 'Overview of trigger IO system'.

param arg_0
OUTPut | OMODe | DIO OUTPut Trigger source is from the output channel. OMODe Trigger source is from the different modes (CC, CR, CV, Sink, OVP, OCP, OPP and

OTP) detected from the output channel. DIO Trigger source is from DIO connector at the instrument rear panel.

Subgroups

6.21.6.1.1.1 Dio

class DioCls

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

Subgroups

6.21.6.1.1.2 Channel

SCPI Commands

```
TRIGger:SEQuence:IMMediate:SOURce:DIO:CHANnel
```

class ChannelCls

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

get(*arg_0: int*) → int

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:DIO:CHANnel
value: int = driver.trigger.sequence.immediate.source.dio.channel.get(arg_0 = 1)
```

Sets or queries the device channel for trigger source 'Digital In Channel'. See Figure 'Overview of trigger IO system'.

param arg_0
OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

return
arg_0: OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

set(*arg_0: int*) → None

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:DIO:CHANnel
driver.trigger.sequence.immediate.source.dio.channel.set(arg_0 = 1)
```

Sets or queries the device channel for trigger source 'Digital In Channel'. See Figure 'Overview of trigger IO system'.

param arg_0
OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

6.21.6.1.1.3 Pin

SCPI Commands

```
TRIGger:SEQuence:IMMediate:SOURce:DIO:PIN
```

class PinCls

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

get(*arg_0: TriggerDioSource*) → TriggerDioSource

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:DIO:PIN
value: enums.TriggerDioSource = driver.trigger.sequence.immediate.source.dio.
↪pin.get(arg_0 = enums.TriggerDioSource.EXT)
```

Sets or queries the DIO pin to trigger on for trigger source 'Digital In Channel'. See Figure 'Overview of trigger IO system'.

param arg_0
IN | EXT IN Pin 3 of DIO connector is monitored. EXT Pin 2 (Ch1) of DIO connector is monitored.

return
arg_0: IN | EXT IN Pin 3 of DIO connector is monitored. EXT Pin 2 (Ch1) of DIO connector is monitored.

set(*arg_0: TriggerDioSource*) → None

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:DIO:PIN
driver.trigger.sequence.immediate.source.dio.pin.set(arg_0 = enums.
↪TriggerDioSource.EXT)
```

Sets or queries the DIO pin to trigger on for trigger source 'Digital In Channel'. See Figure 'Overview of trigger IO system'.

param arg_0
IN | EXT IN Pin 3 of DIO connector is monitored. EXT Pin 2 (Ch1) of DIO connector is monitored.

6.21.6.1.1.4 Omode

SCPI Commands

```
TRIGger:SEQuence:IMMediate:SOURce:OMODe
```

class OmodeCls

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

get(*arg_0: TriggerOperMode*) → TriggerOperMode

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:OMODe
value: enums.TriggerOperMode = driver.trigger.sequence.immediate.source.omode.
↪get(arg_0 = enums.TriggerOperMode.CC)
```

Sets or queries the operation mode to trigger on for trigger source 'operation mode' See Figure 'Overview of trigger IO system'.

param arg_0
CC | CV | CR | SINK | PROTection CC If respective channel operation mode is detected in CC mode, corresponding trigger-out parameters are triggered. CV If respective channel operation mode is detected in CV mode, corresponding trigger-out parameters are triggered. CR If respective channel operation mode is detected in CR mode, corresponding trigger-out parameters are triggered. SINK If respective channel operation mode is detected in sink mode, corresponding trigger-out parameters are triggered. PROTection If respective channel operation mode is detected in protection mode (OVP, OCP, OPP OTP) , corresponding trigger-out parameters are triggered.

return
arg_0: CC | CV | CR | SINK | PROTection CC If respective channel operation mode is detected in CC mode, corresponding trigger-out parameters are triggered. CV If respective channel operation mode is detected in CV mode, corresponding trigger-out parameters are triggered. CR If respective channel operation mode is detected in CR mode, corresponding trigger-out parameters are triggered. SINK If respective channel operation mode is detected in sink mode, corresponding trigger-out parameters are triggered. PROTection If respective channel operation mode is detected in protection mode (OVP, OCP, OPP OTP) , corresponding trigger-out parameters are triggered.

`set`(*arg_0: TriggerOperMode*) → None

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:OMODe
driver.trigger.sequence.immediate.source.omode.set(arg_0 = enums.
↪TriggerOperMode.CC)
```

Sets or queries the operation mode to trigger on for trigger source 'operation mode' See Figure 'Overview of trigger IO system'.

param arg_0
CC | CV | CR | SINK | PROTection CC If respective channel operation mode is detected in CC mode, corresponding trigger-out parameters are triggered. CV If respective channel operation mode is detected in CV mode, corresponding trigger-out parameters are triggered. CR If respective channel operation mode is detected in CR mode, corresponding trigger-out parameters are triggered. SINK If respective channel operation mode is detected in sink mode, corresponding trigger-out parameters are triggered. PROTection If respective channel operation mode is detected in protection mode (OVP, OCP, OPP OTP) , corresponding trigger-out parameters are triggered.

Subgroups

6.21.6.1.1.5 Channel

SCPI Commands

```
TRIGger:SEQuence:IMMediate:SOURce:OMODe:CHANnel
```

`class ChannelCls`

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

get(*arg_0: int*) → int

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:OMODe:CHANnel
value: int = driver.trigger.sequence.immediate.source.omode.channel.get(arg_0 =␣
↪1)
```

Sets or queries the device channel for trigger source 'Operation Modes'. See Figure 'Overview of trigger IO system'.

param arg_0
OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

return
arg_0: OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

set(*arg_0: int*) → None

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:OMODe:CHANnel
driver.trigger.sequence.immediate.source.omode.channel.set(arg_0 = 1)
```

Sets or queries the device channel for trigger source 'Operation Modes'. See Figure 'Overview of trigger IO system'.

param arg_0
OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

6.21.6.1.1.6 Output

class OutputCls

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

Subgroups

6.21.6.1.1.7 Channel

SCPI Commands

```
TRIGger:SEQuence:IMMediate:SOURce:OUTPut:CHANnel
```

class ChannelCls

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

get(*arg_0: int*) → int

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:OUTPut:CHANnel
value: int = driver.trigger.sequence.immediate.source.output.channel.get(arg_0␣
↪= 1)
```

Sets or queries the device channel for trigger source 'Output'. See Figure 'Overview of trigger IO system'.

param arg_0
OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

return
arg_0: OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

set(*arg_0: int*) → None

```
# SCPI: TRIGger[:SEQuence][:IMMediate]:SOURce:OUTPut:CHANnel
driver.trigger.sequence.immediate.source.output.channel.set(arg_0 = 1)
```

Sets or queries the device channel for trigger source 'Output'. See Figure 'Overview of trigger IO system'.

param arg_0
OUT1 | OUTP1 | OUTPut1 | OUT2 | OUTP2 | OUTPut2 OUT1 | OUTP1 | OUTPut1 Ch1 is selected as the device channel for trigger source. OUT2 | OUTP2 | OUTPut2 Ch2 is selected as the device channel for trigger source.

6.21.7 State

SCPI Commands

```
TRIGger:STATe
```

class StateCls

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

get() → bool

```
# SCPI: TRIGger[:STATe]
value: bool = driver.trigger.state.get()
```

Enables or disables the trigger system. Upon being triggered, the selected trigger source method RsNgx.Trigger.Sequence. Immediate.Source.set becomes active. See Figure 'Overview of trigger IO system'.

return
arg_0: 1 Enables the trigger system. 0 Disables the trigger system.

set(*arg_0: bool*) → None

```
# SCPI: TRIGger[:STATe]
driver.trigger.state.set(arg_0 = False)
```

Enables or disables the trigger system. Upon being triggered, the selected trigger source method RsNgx.Trigger.Sequence. Immediate.Source.set becomes active. See Figure 'Overview of trigger IO system'.

param arg_0
1 Enables the trigger system. 0 Disables the trigger system.

CHAPTER

SEVEN

RSNGX UTILITIES

class Utilities

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

Access snippet: `utils = RsNgx.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsNgx.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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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: Optional[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

get_lock() → RLock

Returns the thread lock for the current session.

By default:

- If you create standard new RsNgx instance with new VISA session, the session gets a new thread lock. You can assign it to other RsNgx sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsNgx 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

RSNGX LOGGER

Check the usage in the Getting Started chapter *here*.

class ScpiLogger

Base class for SCPI logging

mode

Sets / returns the Logging mode.

Data Type
LoggingMode

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: Optional[bool] = None*, *udp_log: Optional[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: Optional[bool] = None*, *udp_log: Optional[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

RSNGX 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

D

E

F

G

H

I

L

M

O

R

S

T

U