RsCmwBase

Release 4.0.110.49

Rohde & Schwarz

Apr 16, 2024

CONTENTS:


CMW

CHAPTER

ONE

REVISION HISTORY

1.1 RsCmwBase

Rohde & Schwarz CMW Base System RsCmwBase instrument driver.

Basic Hello-World code:

```
from RsCmwBase import *

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

Supported instruments: CMW500, CMW100, CMW270, CMW280

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

Documentation: https://RsCmwBase.readthedocs.io/

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

1.1.1 Version history

Release Notes for the whole RsCmwXXX group:

Latest release notes summary: Update for FW version 4.0.x

Version 4.0.110.49

- Update for FW version 4.0.x

Version 3.8.xx2

- Fixed several misspelled arguments and command headers

Version 3.8.xx1

- Bluetooth and WLAN update for FW versions 3.8.xxx

Version 3.7.xx8

- Added documentation on ReadTheDocs

Version 3.7.xx7

- Added 3G measurement subsystems RsCmwGsmMeas, RsCmwCdma2kMeas, RsCmwEvdoMeas, RsCmwWcdmaMeas
- Added new data types for commands accepting numbers or ON/OFF:
- int or bool
- float or bool

Version 3.7.xx6

- Added new UDF integer number recognition

Version 3.7.xx5

- Added RsCmwDau

Version 3.7.xx4

- Fixed several interface names
- New release for CMW Base 3.7.90
- New release for CMW Bluetooth 3.7.90

Version 3.7.xx3

- Second release of the CMW python drivers packet
- New core component RsInstrument
- Previously, the groups starting with CATalog: e.g. 'CATalog:SIGNaling:TOPology:PLMN' were reordered to 'SIGNaling:TOPology:PLMN:CATALOG' give more contextual meaning to the method/property name. This is now reverted back, since it was hard to find the desired functionality.
- Reorganized Utilities interface to sub-groups

Version 3.7.xx2

- Fixed some misspeling errors
- Changed enum and repCap types names
- All the assemblies are signed with Rohde & Schwarz signature

Version 1.0.0.0

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsCmwBase 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 RsCmwBase example:

```
""" Example on how to use the python RsCmw auto-generated instrument driver showing:
- usage of basic properties of the cmw_base object
- basic concept of setting commands and repcaps: DISPlay:WINDow<n>:SELect
- cmw_xxx drivers reliability interface usage
"""

from RsCmwBase import *  # install from pypi.org

RsCmwBase.assert_minimum_version('3.7.90.38')
cmw_base = RsCmwBase('TCPIP::10.112.1.116::INSTR', True, False)
print(f'CMW Base IND: {cmw_base.utilities.idn_string}')
print(f'CMW Instrument options:\n{",".join(cmw_base.utilities.instrument_options)}')
cmw_base.utilities.visa_timeout = 5000

# Sends OPC after each command
cmw_base.utilities.opc_query_after_write = False
```

(continues on next page)

(continued from previous page)

```
# Checks for syst:err? after each command / query
cmw_base.utilities.instrument_status_checking = True

# DISPlay:WINDow<n>:SELect
cmw_base.display.window.select.set(repcap.Window.Win1)
cmw_base.display.window.repcap_window_set(repcap.Window.Win2)
cmw_base.display.window.select.set()

# Self-test
self_test = cmw_base.utilities.self_test()
print(f'CMW self-test result: {self_test} - {"Passed" if self_test[0] == 0 else "Failed"}
↪"')

# Driver's Interface reliability offers a convenient way of reacting on the return value␣
↪Reliability Indicator
cmw_base.reliability.ExceptionOnError = True


# Callback to use for the reliability indicator update event
def my_reliability_handler(event_args: ReliabilityEventArgs):
        print(f'Base Reliability updated.\nContext: {event_args.context}\nMessage:
↪{event_args.message}')


# We register a callback for each change in the reliability indicator
cmw_base.reliability.on_update_handler = my_reliability_handler

# You can obtain the last value of the returned reliability
print(f"\nReliability last value: {cmw_base.reliability.last_value}, context '{cmw_base.
↪reliability.last_context}', message: {cmw_base.reliability.last_message}")

# Reference Frequency Source
cmw_base.system.reference.frequency.set_source(enums.SourceIntExt.INTernal)

# Close the session
cmw_base.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

RsCmwBase 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 RsCmwBase`

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 `RsCmwBase` 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 RsCmwBase offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwBase needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwBase package to your computer from the pypi.org: https://pypi.org/project/RsCmwBase/#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\RsCmwBase-4.0.110.49.tar`

2.3 Finding Available Instruments

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

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

from RsCmwBase import *

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

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

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

- Installed RsCmwBase Python module Version 4.0.110 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCmwBase import *

# A good practice is to assure that you have a certain minimum version installed
RsCmwBase.assert_minimum_version('4.0.110')
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 = RsCmwBase(resource_string_1)

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

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

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

Do not care about specialty of each session kind; RsCmwBase 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 = RsCmwBase('TCPIP::192.168.56.101::hislip0', id_query=True, reset=True)
```

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

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

from RsCmwBase import *

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

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

from RsCmwBase import *

driver = RsCmwBase('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}'")
```

(continues on next page)

(continued from previous page)

```
# 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 = RsCmwBase('TCPIP::192.168.56.101::hislip0', True, True, "Simulate=True")
```

More `option_string` tokens are separated by comma:

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

Shared Session

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

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

from RsCmwBase import *

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

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

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

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

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

from RsCmwBase import *

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

(continues on next page)

(continued from previous page)

```
# 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 RsCmwBase raises an exception. Speaking of exceptions, an important feature of the RsCmwBase 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 RsCmwBase import *

driver = RsCmwBase('TCPIP::192.168.56.101::INSTR')
driver.utilities.visa_timeout = 5000
driver.utilities.instrument_status_checking = True
driver.utilities.write_int('SWEEP:COUNT ', 10)  # sending 'SWEEP:COUNT 10'
driver.utilities.write_bool('SOURCE:RF:OUTPUT:STATE ', True)  # sending
↪'SOURCE:RF:OUTPUT:STATE ON'
driver.utilities.write_float('SOURCE:RF:FREQUENCY ', 1E9)  # sending 'SOURCE:RF:FREQUENCY␣
↪1000000000'

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

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

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

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

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

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

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

2.6 Error Checking

RsCmwBase 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 RsCmwBase 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 RsCmwBase import *

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

(continues on next page)

(continued from previous page)

```
    driver = RsCmwBase('TCPIP::10.112.1.179::hislip0')
except ResourceError as e:
    print(e.args[0])
    print('Your instrument is probably OFF...')
    # Exit now, no point of continuing
    exit(1)

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

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

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

RsCmwBase 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 RsCmwBase 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 = RsCmwBase('TCPIP::192.168.56.101::INSTR')

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

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

If you start it, you might wonder (or maybe not): why is the `args.total_size = None`? The reason is, in this particular case the RsCmwBase 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, RsCmwBase 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 RsCmwBase object
"""

import threading
from RsCmwBase import *


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


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

(continues on next page)

(continued from previous page)

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsCmwBase import *


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

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

# Wait for all threads to join this main thread
for t in threads:
    t.join()
```

(continues on next page)

(continued from previous page)

```
print('All threads ended')

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

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

Multiple instrument sessions accessed from multiple threads

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

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

Run the following example:

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

import threading
from RsCmwBase import *


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

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

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

(continues on next page)

(continued from previous page)

```
    t.start()
    threads.append(t)
print('All threads started')

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

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

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

2.12 Logging

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

What can you actually do with the logger?

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

Let us take this basic example:

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

from RsCmwBase import *

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

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

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

Console output:

```
10:29:10.819     TCPIP::192.168.1.101::INSTR     0.976 ms  Write: *RST
10:29:10.819     TCPIP::192.168.1.101::INSTR  1884.985 ms  Status check: OK
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

See the full logger help *here*.

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

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

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

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

from RsCmwBase import *

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

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

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

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

# Custom user entry
```

(continues on next page)

(continued from previous page)

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

driver.utilities.reset()

# Close the session
driver.close()

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

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

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

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

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

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

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

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

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

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

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

from RsCmwBase import *

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

Console output:

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

Notice the following:

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

CHAPTER

THREE

ENUMS

3.1 AdjustStatus

```
# Example value:
value = enums.AdjustStatus.ADJust
# All values (2x):
ADJust | NADJust
```

3.2 BaseAdjState

```
# Example value:
value = enums.BaseAdjState.ADJusted
# All values (8x):
ADJusted | AUTonomous | COUPled | INValid | OFF | ON | PENDing | RDY
```

3.3 BoxNumber

```
# First value:
value = enums.BoxNumber.BOX1
# Last value:
value = enums.BoxNumber.NAV
# All values (9x):
BOX1 | BOX2 | BOX3 | BOX4 | BOX5 | BOX6 | BOX7 | BOX8
NAV
```

3.4 ByteOrder

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

3.5 CatalogFormat

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

3.6 CmwCurrentStatus

```
# Example value:
value = enums.CmwCurrentStatus.ERRor
# All values (6x):
ERRor | MCCNconnected | MCMW | PCINconnected | SALone | STBY
```

3.7 CmwMode

```
# Example value:
value = enums.CmwMode.GENerator
# All values (3x):
GENerator | LISTener | STANdalone
```

3.8 CmwSetStatus

```
# Example value:
value = enums.CmwSetStatus.MCMW
# All values (3x):
MCMW | SALone | STBY
```

3.9 ColorSet

```
# Example value:
value = enums.ColorSet.DEF
# All values (1x):
DEF
```

3.10 CorrResult

```
# Example value:
value = enums.CorrResult.FAIL
# All values (4x):
FAIL | IPR | NCAP | PASS
```

3.11 DataFormat

```
# Example value:
value = enums.DataFormat.ASCii
# All values (8x):
ASCii | BINary | HEXadecimal | INTeger | OCTal | PACKed | REAL | UINTeger
```

3.12 DefaultUnitAngle

```
# Example value:
value = enums.DefaultUnitAngle.DEG
# All values (3x):
DEG | GRAD | RAD
```

3.13 DefaultUnitCapacity

```
# First value:
value = enums.DefaultUnitCapacity.AF
# Last value:
value = enums.DefaultUnitCapacity.UF
# All values (13x):
AF | EXF | F | FF | GF | KF | MF | MIF
NF | PEF | PF | TF | UF
```

3.14 DefaultUnitCharge

```
# First value:
value = enums.DefaultUnitCharge.AC
# Last value:
value = enums.DefaultUnitCharge.UC
# All values (13x):
AC | C | EXC | FC | GC | KC | MC | MIC
NC | PC | PEC | TC | UC
```

3.15 DefaultUnitConductance

```
# First value:
value = enums.DefaultUnitConductance.ASIE
# Last value:
value = enums.DefaultUnitConductance.USIE
# All values (13x):
ASIE | EXSie | FSIE | GSIE | KSIE | MISie | MSIE | NSIE
PESie | PSIE | SIE | TSIE | USIE
```

3.16 DefaultUnitCurrent

```
# First value:
value = enums.DefaultUnitCurrent.A
# Last value:
value = enums.DefaultUnitCurrent.UA
# All values (18x):
A | AA | DBA | DBMA | DBNA | DBPA | DBUA | EXA
FA | GA | KA | MA | MAA | NA | PA | PEA
TA | UA
```

3.17 DefaultUnitEnergy

```
# First value:
value = enums.DefaultUnitEnergy.AJ
# Last value:
value = enums.DefaultUnitEnergy.UJ
# All values (13x):
AJ | EXJ | FJ | GJ | J | KJ | MIJ | MJ
NJ | PEJ | PJ | TJ | UJ
```

3.18 DefaultUnitFrequency

```
# First value:
value = enums.DefaultUnitFrequency.AHZ
# Last value:
value = enums.DefaultUnitFrequency.UHZ
# All values (13x):
AHZ | EXHZ | FHZ | GHZ | HZ | KHZ | MHZ | MIHZ
NHZ | PEHZ | PHZ | THZ | UHZ
```

3.19 DefaultUnitLenght

```
# First value:
value = enums.DefaultUnitLenght.AM
# Last value:
value = enums.DefaultUnitLenght.UM
# All values (13x):
AM | EXM | FM | GM | KM | M | MAM | MM
NM | PEM | PM | TM | UM
```

3.20 DefaultUnitPower

```
# First value:
value = enums.DefaultUnitPower.AW
# Last value:
value = enums.DefaultUnitPower.W
# All values (19x):
AW | DBC | DBMW | DBNW | DBPW | DBUW | DBW | EXW
FW | GW | KW | MIW | MW | NW | PEW | PW
TW | UW | W
```

3.21 DefaultUnitResistor

```
# First value:
value = enums.DefaultUnitResistor.AOHM
# Last value:
value = enums.DefaultUnitResistor.UOHM
# All values (13x):
AOHM | EXOHm | FOHM | GOHM | KOHM | MIOHm | MOHM | NOHM
OHM | PEOHm | POHM | TOHM | UOHM
```

3.22 DefaultUnitTemperature

```
# Example value:
value = enums.DefaultUnitTemperature.C
# All values (6x):
C | CEL | F | FAR | K | KEL
```

3.23 DefaultUnitTime

```
# First value:
value = enums.DefaultUnitTime.AS
# Last value:
value = enums.DefaultUnitTime.US
# All values (18x):
AS | EXS | FS | GS | H | HOUR | KS | M
MAS | MIN | MS | NS | PES | PS | S | SEC
TS | US
```

3.24 DefaultUnitVoltage

```
# First value:
value = enums.DefaultUnitVoltage.AV
# Last value:
value = enums.DefaultUnitVoltage.V
# All values (18x):
AV | DBMV | DBNV | DBPV | DBUV | DBV | EXV | FV
GV | KV | MAV | MV | NV | PEV | PV | TV
UV | V
```

3.25 DiagLoggigMode

```
# Example value:
value = enums.DiagLoggigMode.DETailed
# All values (3x):
DETailed | OFF | SIMPle
```

3.26 DiagLoggingDevice

```
# Example value:
value = enums.DiagLoggingDevice.ALL
# All values (3x):
ALL | DEBug | MEMory
```

3.27 DirectionIo

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

3.28 DisplayLanguage

```
# First value:
value = enums.DisplayLanguage.AR
# Last value:
value = enums.DisplayLanguage.ZH
# All values (14x):
AR | CS | DA | DE | EN | ES | FR | IT
JA | KO | RU | SV | TR | ZH
```

3.29 DisplayMode

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

3.30 DisplayStrategy

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

3.31 ExpertSetup

```
# First value:
value = enums.ExpertSetup.BBG1
# Last value:
value = enums.ExpertSetup.SUW7
# All values (101x):
BBG1 | BBG2 | BBG3 | BBG4 | BBG5 | BBG6 | BBG7 | BBM1
BBM2 | BBM3 | BBM4 | BBM5 | BBM6 | BBM7 | INValid | PANY
PI1 | PI2 | PO1 | PO2 | R11 | R118 | R11Ci | R11Co
R11O | R12 | R12Ci | R12Co | R13 | R13Ci | R13Co | R13O
R14 | R14Ci | R14Co | R15 | R16 | R17 | R18 | R1CI
R1CO | R1O | R21Ci | R21Co | R21O | R22Ci | R22Co | R23Ci
R23Co | R23O | R24Ci | R24Co | R2CI | R2CO | R31Ci | R31Co
R31O | R32Ci | R32Co | R33Ci | R33Co | R33O | R34Ci | R34Co
R3CI | R3CO | R3O | R41Ci | R41Co | R41O | R42Ci | R42Co
R43Ci | R43Co | R43O | R44Ci | R44Co | R4CI | R4CO | RRX1
RRX2 | RRX3 | RRX4 | RTX1 | RTX2 | RTX3 | RTX4 | SUU1
SUU2 | SUU3 | SUU4 | SUU5 | SUU6 | SUU7 | SUW1 | SUW2
SUW3 | SUW4 | SUW5 | SUW6 | SUW7
```

3.32 ExpressionMode

```
# Example value:
value = enums.ExpressionMode.REGex
# All values (2x):
REGex | STRing
```

3.33 FanMode

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

3.34 FontType

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

3.35 JoinAction

```
# Example value:
value = enums.JoinAction.CTASk
# All values (3x):
CTASk | DONE | STASk
```

3.36 MutexAction

```
# Example value:
value = enums.MutexAction.DONothing
# All values (2x):
DONothing | RELock
```

3.37 MutexState

```
# Example value:
value = enums.MutexState.LOCKed
# All values (3x):
LOCKed | NEWLocked | UNLocked
```

3.38 OperationMode

```
# Example value:
value = enums.OperationMode.LOCal
# All values (2x):
LOCal | REMote
```

3.39 OscillatorType

```
# Example value:
value = enums.OscillatorType.OCXO
# All values (2x):
OCXO | TCXO
```

3.40 ProductType

```
# Example value:
value = enums.ProductType.ALL
# All values (5x):
ALL | FWA | HWOPtion | SWOPtion | SWPackage
```

3.41 RemoteTraceEnable

```
# Example value:
value = enums.RemoteTraceEnable.ANALysis
# All values (4x):
ANALysis | LIVE | OFF | ON
```

3.42 RemoteTraceFileFormat

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

3.43 RemoteTraceStartMode

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

3.44 RemoteTraceStopMode

```
# Example value:
value = enums.RemoteTraceStopMode.AUTO
# All values (4x):
AUTO | BUFFerfull | ERRor | EXPLicit
```

3.45 ResourceState

```
# Example value:
value = enums.ResourceState.ACTive
# All values (8x):
ACTive | ADJusted | INValid | OFF | PENDing | QUEued | RDY | RUN
```

3.46 RfConverterInPath

```
# Example value:
value = enums.RfConverterInPath.RF1
# All values (4x):
RF1 | RF2 | RF3 | RF4
```

3.47 RollkeyMode

```
# Example value:
value = enums.RollkeyMode.CURSors
# All values (3x):
CURSors | VERTical | ZIGZag
```

3.48 RxTxDirection

```
# Example value:
value = enums.RxTxDirection.RX
# All values (3x):
RX | RXTX | TX
```

3.49 ScreenshotFormat

```
# Example value:
value = enums.ScreenshotFormat.BMP
# All values (3x):
BMP | JPG | PNG
```

3.50 Segment

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

3.51 SignalSlope

```
# Example value:
value = enums.SignalSlope.FEDGe
# All values (2x):
FEDGe | REDGe
```

3.52 SocketProtocol

```
# Example value:
value = enums.SocketProtocol.AGILent
# All values (3x):
AGILent | IEEE1174 | RAW
```

3.53 SourceIntExt

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

3.54 StatRegFormat

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

3.55 StoragePlace

```
# Example value:
value = enums.StoragePlace.EEPRom
# All values (3x):
EEPRom | FILE | SIM
```

3.56 SubnetScope

```
# Example value:
value = enums.SubnetScope.ALL
# All values (4x):
ALL | DALL | DEXTern | EXTern
```

3.57 SyncPolling

```
# Example value:
value = enums.SyncPolling.NPOLling
# All values (2x):
NPOLling | POLLing
```

3.58 SyncResult

```
# Example value:
value = enums.SyncResult.DSTask
# All values (5x):
DSTask | NRDY | NSTask | RDY | TOUT
```

3.59 TextFormatting

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

3.60 Type

```
# Example value:
value = enums.Type.CALibration
# All values (4x):
CALibration | FSCorrection | OGCal | UCORrection
```

3.61 UserRole

```
# Example value:
value = enums.UserRole.ADMin
# All values (5x):
ADMin | DEVeloper | SERVice | UEXTended | USER
```

3.62 ValidityScope

```
# Example value:
value = enums.ValidityScope.ALL
# All values (4x):
ALL | CLICense | FUNCtional | VALid
```

3.63 ValidityScopeA

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

3.64 ValidityScopeB

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

CHAPTER

FOUR

REPCAPS

4.1 BitNr

```
# First value:
value = repcap.BitNr.Nr8
# Range:
Nr8 .. Nr12
# All values (5x):
Nr8 | Nr9 | Nr10 | Nr11 | Nr12
```

4.2 CmwVariant

```
# First value:
value = repcap.CmwVariant.Cmw1
# Values (2x):
Cmw1 | Cmw100
```

4.3 Eout

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

4.4 FileNr

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

4.5 Frequency

```
# First value:
value = repcap.Frequency.Freq1
# Values (4x):
Freq1 | Freq2 | Freq3 | Freq4
```

4.6 GpibInstance

```
# First value:
value = repcap.GpibInstance.Inst1
# Range:
Inst1 .. Inst32
# All values (32x):
Inst1 | Inst2 | Inst3 | Inst4 | Inst5 | Inst6 | Inst7 | Inst8
Inst9 | Inst10 | Inst11 | Inst12 | Inst13 | Inst14 | Inst15 | Inst16
Inst17 | Inst18 | Inst19 | Inst20 | Inst21 | Inst22 | Inst23 | Inst24
Inst25 | Inst26 | Inst27 | Inst28 | Inst29 | Inst30 | Inst31 | Inst32
```

4.7 HislipInstance

```
# First value:
value = repcap.HislipInstance.Inst1
# Range:
Inst1 .. Inst32
# All values (32x):
Inst1 | Inst2 | Inst3 | Inst4 | Inst5 | Inst6 | Inst7 | Inst8
Inst9 | Inst10 | Inst11 | Inst12 | Inst13 | Inst14 | Inst15 | Inst16
Inst17 | Inst18 | Inst19 | Inst20 | Inst21 | Inst22 | Inst23 | Inst24
Inst25 | Inst26 | Inst27 | Inst28 | Inst29 | Inst30 | Inst31 | Inst32
```

4.8 IpAddress

```
# First value:
value = repcap.IpAddress.Addr1
# Values (3x):
Addr1 | Addr2 | Addr3
```

4.9 NwAdapter

```
# First value:
value = repcap.NwAdapter.Adapter1
# Range:
Adapter1 .. Adapter5
# All values (5x):
Adapter1 | Adapter2 | Adapter3 | Adapter4 | Adapter5
```

4.10 RsibInstance

```
# First value:
value = repcap.RsibInstance.Inst1
# Range:
Inst1 .. Inst32
# All values (32x):
Inst1 | Inst2 | Inst3 | Inst4 | Inst5 | Inst6 | Inst7 | Inst8
Inst9 | Inst10 | Inst11 | Inst12 | Inst13 | Inst14 | Inst15 | Inst16
Inst17 | Inst18 | Inst19 | Inst20 | Inst21 | Inst22 | Inst23 | Inst24
Inst25 | Inst26 | Inst27 | Inst28 | Inst29 | Inst30 | Inst31 | Inst32
```

4.11 RxFilter

```
# First value:
value = repcap.RxFilter.Nr1
# Range:
Nr1 .. Nr10
# All values (10x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10
```

4.12 Slot

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

4.13 SocketInstance

```
# First value:
value = repcap.SocketInstance.Inst1
# Range:
Inst1 .. Inst32
# All values (32x):
Inst1 | Inst2 | Inst3 | Inst4 | Inst5 | Inst6 | Inst7 | Inst8
Inst9 | Inst10 | Inst11 | Inst12 | Inst13 | Inst14 | Inst15 | Inst16
Inst17 | Inst18 | Inst19 | Inst20 | Inst21 | Inst22 | Inst23 | Inst24
Inst25 | Inst26 | Inst27 | Inst28 | Inst29 | Inst30 | Inst31 | Inst32
```

4.14 Trigger

```
# First value:
value = repcap.Trigger.Trg1
# Values (4x):
Trg1 | Trg2 | Trg3 | Trg4
```

4.15 TxFilter

```
# First value:
value = repcap.TxFilter.Nr1
# Range:
Nr1 .. Nr10
# All values (10x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10
```

4.16 VxiInstance

```
# First value:
value = repcap.VxiInstance.Inst1
# Range:
Inst1 .. Inst32
# All values (32x):
Inst1 | Inst2 | Inst3 | Inst4 | Inst5 | Inst6 | Inst7 | Inst8
Inst9 | Inst10 | Inst11 | Inst12 | Inst13 | Inst14 | Inst15 | Inst16
Inst17 | Inst18 | Inst19 | Inst20 | Inst21 | Inst22 | Inst23 | Inst24
Inst25 | Inst26 | Inst27 | Inst28 | Inst29 | Inst30 | Inst31 | Inst32
```

4.17 Window

```
# First value:
value = repcap.Window.Win1
# Range:
Win1 .. Win32
# All values (32x):
Win1 | Win2 | Win3 | Win4 | Win5 | Win6 | Win7 | Win8
Win9 | Win10 | Win11 | Win12 | Win13 | Win14 | Win15 | Win16
Win17 | Win18 | Win19 | Win20 | Win21 | Win22 | Win23 | Win24
Win25 | Win26 | Win27 | Win28 | Win29 | Win30 | Win31 | Win32
```

CHAPTER

FIVE

EXAMPLES

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

```
""" Example on how to use the python RsCmw auto-generated instrument driver showing:
- usage of basic properties of the cmw_base object
- basic concept of setting commands and repcaps: DISPlay:WINDow<n>:SELect
- cmw_xxx drivers reliability interface usage
"""

from RsCmwBase import *  # install from pypi.org

RsCmwBase.assert_minimum_version('3.7.90.38')
cmw_base = RsCmwBase('TCPIP::10.112.1.116::INSTR', True, False)
print(f'CMW Base IND: {cmw_base.utilities.idn_string}')
print(f'CMW Instrument options:\n{",".join(cmw_base.utilities.instrument_options)}')
cmw_base.utilities.visa_timeout = 5000

# Sends OPC after each command
cmw_base.utilities.opc_query_after_write = False

# Checks for syst:err? after each command / query
cmw_base.utilities.instrument_status_checking = True

# DISPlay:WINDow<n>:SELect
cmw_base.display.window.select.set(repcap.Window.Win1)
cmw_base.display.window.repcap_window_set(repcap.Window.Win2)
cmw_base.display.window.select.set()

# Self-test
self_test = cmw_base.utilities.self_test()
print(f'CMW self-test result: {self_test} - {"Passed" if self_test[0] == 0 else "Failed"}
↪"')

# Driver's Interface reliability offers a convenient way of reacting on the return value␣
↪Reliability Indicator
cmw_base.reliability.ExceptionOnError = True


# Callback to use for the reliability indicator update event
def my_reliability_handler(event_args: ReliabilityEventArgs):
        print(f'Base Reliability updated.\nContext: {event_args.context}\nMessage:
```

(continues on next page)

(continued from previous page)

```
↪{event_args.message}')


# We register a callback for each change in the reliability indicator
cmw_base.reliability.on_update_handler = my_reliability_handler

# You can obtain the last value of the returned reliability
print(f"\nReliability last value: {cmw_base.reliability.last_value}, context '{cmw_base.
↪reliability.last_context}', message: {cmw_base.reliability.last_message}")

# Reference Frequency Source
cmw_base.system.reference.frequency.set_source(enums.SourceIntExt.INTernal)

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

CHAPTER

SIX

RSCMWBASE API STRUCTURE

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

409 total commands, 32 Subgroups, 3 group commands

Initializes new RsCmwBase session.

Parameter options tokens examples:

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

- `ScpiQuotes = double'. - for SCPI commands, you can define how strings are quoted. With single or double quotes. Possible values: single | double | {char}. Default: ``single`
- `LoggingMode = On` - Sets the logging status right from the start. Default: `Off`
- `LoggingName = 'MyDevice'` - Sets the name to represent the session in the log entries. Default: `'resource_name'`
- `LogToGlobalTarget = True` - Sets the logging target to the class-property previously set with RsCmwBase.set_global_logging_target() Default: `False`
- `LoggingToConsole = True` - Immediately starts logging to the console. Default: False
- `LoggingToUdp = True` - Immediately starts logging to the UDP port. Default: False
- `LoggingUdpPort = 49200` - UDP port to log to. Default: 49200

Parameters

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

static assert_minimum_version(*min_version: str*) → None

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

classmethod clear_global_logging_relative_timestamp() → None

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

close() → None

Closes the active RsCmwBase session.

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

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

Parameters

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

get_device_number() → int

```
# SCPI: *DEV
value: int = driver.get_device_number()
```

Queries the device number. It equals the 'Assigned Instrument' number minus 1.

return
instrument_no: integer Range: 0 to n

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_global_opc() → bool

```
# SCPI: *GOPC
value: bool = driver.get_global_opc()
```

No command help available

return
gopc: No help available

get_macro_enable() → bool

```
# SCPI: *EMC
value: bool = driver.get_macro_enable()
```

Enables or disables the execution of all macros that are defined for the active remote connection. Note: In contrast to SCPI specifications, macro execution is disabled by default.

return
enable: No help available

get_session_handle() → object

Returns the underlying session handle.

get_total_execution_time() → timedelta

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

get_total_time() → timedelta

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

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

Finds all the resources defined by the expression

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

Parameters

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

reset_time_statistics() → None

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

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

set_device_number(*instrument_no: int*) → None

```
# SCPI: *DEV
driver.set_device_number(instrument_no = 1)
```

Queries the device number. It equals the 'Assigned Instrument' number minus 1.

param instrument_no
No help available

classmethod set_global_logging_relative_timestamp(*timestamp: datetime*) → None

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

classmethod set_global_logging_relative_timestamp_now() → None

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

classmethod set_global_logging_target(*target*) → None

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

set_macro_enable(*enable: bool*) → None

```
# SCPI: *EMC
driver.set_macro_enable(enable = False)
```

Enables or disables the execution of all macros that are defined for the active remote connection. Note: In contrast to SCPI specifications, macro execution is disabled by default.

param enable
ON | OFF | 1 | 0 Boolean value to enable or disable macro execution. In the disabled state (OFF / 0) , macros in a command sequence are not expanded. The R&S CMW issues an error message: 113, Undefined header;MacroLabel.

Subgroups

6.1 Buffer

SCPI Commands :

```
STARt:BASE:BUFFer
STOP:BASE:BUFFer
CONTinue:BASE:BUFFer
DELete:BASE:BUFFer
CLEar:BASE:BUFFer
FETCh:BASE:BUFFer
```

class BufferCls

Buffer commands group definition. 7 total commands, 1 Subgroups, 6 group commands

clear(*buffer: str*) → None

```
# SCPI: CLEar:BASE:BUFFer
driver.buffer.clear(buffer = 'abc')
```

Clears the contents of a buffer. You get an empty buffer that you can fill with new commands.

param buffer
string

continue_py(*buffer: str*) → None

```
# SCPI: CONTinue:BASE:BUFFer
driver.buffer.continue_py(buffer = 'abc')
```

Reactivates a buffer which was deactivated via method RsCmwBase.Buffer.stop) . The R&S CMW continues writing data to the buffer.

param buffer
string

delete(*buffer: str*) → None

```
# SCPI: DELete:BASE:BUFFer
driver.buffer.delete(buffer = 'abc')
```

Deletes a buffer.

param buffer
string

fetch(*buffer: str*, *line_number: int*) → str

```
# SCPI: FETCh:BASE:BUFFer
value: str = driver.buffer.fetch(buffer = 'abc', line_number = 1)
```

Reads the contents of a buffer line. Buffer contents are stored line by line. Every query generates a new buffer line. The queries are not stored together with the results. Reading buffer contents is non-destructive. The lines can be read in arbitrary order.

param buffer
No help available

param line_number
integer Line number, selects the line to be read.

return
line: No help available

start(*buffer: str*) → None

```
# SCPI: STARt:BASE:BUFFer
driver.buffer.start(buffer = 'abc')
```

Creates and activates a buffer. If the buffer exists already, it is cleared (equivalent to method RsCmwBase.Buffer.clear) .

param buffer
string The buffer is identified via this label in all buffer commands.

stop() → None

```
# SCPI: STOP:BASE:BUFFer
driver.buffer.stop()
```

Deactivates the active buffer. Only one buffer can be active at a time. The buffer and its contents are maintained, but data recording is paused. Use method RsCmwBase.Buffer.continue_py to reactivate a buffer.

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

```
# SCPI: STOP:BASE:BUFFer
driver.buffer.stop_with_opc()
```

Deactivates the active buffer. Only one buffer can be active at a time. The buffer and its contents are maintained, but data recording is paused. Use method RsCmwBase.Buffer.continue_py to reactivate a buffer.

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

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

Cloning the Group

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

Subgroups

6.1.1 LineCount

SCPI Command :

```
FETCh:BASE:BUFFer:LINecount
```

class LineCountCls

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

fetch(*buffer: str*) → int

```
# SCPI: FETCh:BASE:BUFFer:LINecount
value: int = driver.buffer.lineCount.fetch(buffer = 'abc')
```

Returns the number of lines in a buffer.

param buffer
No help available

return
size: decimal Number of lines in the buffer.

6.2 Calibration

SCPI Commands :

```
CALibration:BASE:ALL
CALibration:BASE:ACFile
```

class CalibrationCls

Calibration commands group definition. 10 total commands, 3 Subgroups, 2 group commands

class AcFileStruct

Structure for reading output parameters. Fields:

- Type_Py: str: No parameter help available
- Date: str: No parameter help available

class AllStruct

Structure for reading output parameters. Fields:

- Date: List[str]: string Date of the calibration
- Time: List[str]: string Time of the calibration
- Type_Py: List[enums.Type]: FSCorrection | UCORrection | CALibration | OGCal Type of the calibration FSCorrection: Correction performed in factory or service UCORrection: Correction performed by a customer CALibration: Verification in the factory OGCal: Verification by the service (outgoing calibration)

get_ac_file() → AcFileStruct

```
# SCPI: CALibration:BASE:ACFile
value: AcFileStruct = driver.calibration.get_ac_file()
```

Query name and creation date of the currently active RF path correction file.

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

get_all() → AllStruct

```
# SCPI: CALibration:BASE:ALL
value: AllStruct = driver.calibration.get_all()
```

Query the stored calibration information. A comma-separated list is returned, containing three parameters per calibration, as described below.

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

Cloning the Group

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

Subgroups

6.2.1 Ipc

SCPI Commands :

```
CALibration:BASE:IPC:RESult
CALibration:BASE:IPC:VALues
CALibration:BASE:IPC:LOG
```

class IpcCls

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

class ResultStruct

Structure for reading output parameters. Fields:

- Result_Number: int: No parameter help available
- Date: str: No parameter help available
- Result_Text: str: No parameter help available

class ValuesStruct

Structure for reading output parameters. Fields:

- Min_Py: List[float]: No parameter help available
- Fmin: List[int]: No parameter help available
- Max_Py: List[float]: No parameter help available
- Fmax: List[int]: No parameter help available

get_log() → List[str]

```
# SCPI: CALibration:BASE:IPC:LOG
value: List[str] = driver.calibration.ipc.get_log()
```

No command help available

return
file_path: No help available

get_result() → ResultStruct

```
# SCPI: CALibration:BASE:IPC:RESult
value: ResultStruct = driver.calibration.ipc.get_result()
```

No command help available

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

get_values() → ValuesStruct

```
# SCPI: CALibration:BASE:IPC:VALues
value: ValuesStruct = driver.calibration.ipc.get_values()
```

No command help available

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

6.2.2 Ipcr

SCPI Commands :

```
CALibration:BASE:IPCR:DATE
CALibration:BASE:IPCR:STATe
CALibration:BASE:IPCR:RESult
```

class IpcrCls

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

get_date() → str

```
# SCPI: CALibration:BASE:IPCR:DATE
value: str = driver.calibration.ipcr.get_date()
```

No command help available

return
date: No help available

get_result() → List[str]

```
# SCPI: CALibration:BASE:IPCR:RESult
value: List[str] = driver.calibration.ipcr.get_result()
```

No command help available

return
result: No help available

get_state() → List[int]

```
# SCPI: CALibration:BASE:IPCR:STATe
value: List[int] = driver.calibration.ipcr.get_state()
```

No command help available

return
state: No help available

6.2.3 Latest

SCPI Command :

```
CALibration:BASE:LATest
```

class LatestCls

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

class GetStruct

Response structure. Fields:

- Date: str: string Date of the calibration
- Time: str: string Time of the calibration
- Type_Py: enums.Type: FSCorrection | UCORrection | CALibration | OGCal Type of the calibration. Can be specified to query the last calibration of a specific type and is returned as last value. FSCorrection: Correction performed in factory or service UCORrection: Correction performed by a customer CALibration: Verification in the factory OGCal: Verification by the service (outgoing calibration)

get(*type_py: Type = None*) → GetStruct

```
# SCPI: CALibration:BASE:LATest
value: GetStruct = driver.calibration.latest.get(type_py = enums.Type.
↪CALibration)
```

Query the stored information about the latest calibration. Optionally, you can specify <Type> to query information about the latest calibration of this type. The information is returned as '<Date>','<Time>',<Type>.

param type_py
FSCorrection | UCORrection | CALibration | OGCal Type of the calibration. Can be specified to query the last calibration of a specific type and is returned as last value. FSCorrection: Correction performed in factory or service UCORrection: Correction performed by a customer CALibration: Verification in the factory OGCal: Verification by the service (outgoing calibration)

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

Cloning the Group

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

Subgroups

6.2.3.1 Specific

SCPI Command :

```
CALibration:BASE:LATest:SPECific
```

class SpecificCls

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

class GetStruct

Response structure. Fields:

- Date: str: string Date of the calibration
- Time: str: string Time of the calibration

get(*mode: Type*) → GetStruct

```
# SCPI: CALibration:BASE:LATest:SPECific
value: GetStruct = driver.calibration.latest.specific.get(mode = enums.Type.
↪CALibration)
```

Query date and time of the latest calibration of the specified type.

param mode
FSCorrection | UCORrection | CALibration | OGCal Type of the calibration for which information is queried. FSCorrection: Correction performed in factory or service UCORrection: Correction performed by a customer CALibration: Verification in the factory OGCal: Verification by the service (outgoing calibration)

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

6.3 Catalog

class CatalogCls

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

Cloning the Group

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

Subgroups

6.3.1 Correction

class CorrectionCls

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

Cloning the Group

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

Subgroups

6.3.1.1 IfEqualizer

SCPI Command :

```
CATalog:BASE:CORRection:IFEQualizer:SNAMe
```

class IfEqualizerCls

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

get_sname() → List[str]

```
# SCPI: CATalog:BASE:CORRection:IFEQualizer:SNAMe
value: List[str] = driver.catalog.correction.ifEqualizer.get_sname()
```

No command help available

return
slot: No help available

Cloning the Group

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

Subgroups

6.3.1.1.1 Slot<Slot>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.catalog.correction.ifEqualizer.slot.repcap_slot_get()
driver.catalog.correction.ifEqualizer.slot.repcap_slot_set(repcap.Slot.Nr1)
```

class SlotCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.catalog.correction.ifEqualizer.slot.clone()
```

Subgroups

6.3.1.1.1.1 RxFilter

SCPI Command :

```
CATalog:BASE:CORRection:IFEQualizer:SLOT<Slot>:RXFilter
```

class RxFilterCls

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

get(*slot=Slot.Default*) → List[str]

```
# SCPI: CATalog:BASE:CORRection:IFEQualizer:SLOT<Slot>:RXFilter
value: List[str] = driver.catalog.correction.ifEqualizer.slot.rxFilter.get(slot␣
↪= repcap.Slot.Default)
```

No command help available

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

return
filter_py: No help available

6.3.1.1.2 TxFilter

SCPI Command :

```
CATalog:BASE:CORRection:IFEQualizer:SLOT<Slot>:TXFilter
```

class TxFilterCls

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

get(*slot=Slot.Default*) → List[str]

```
# SCPI: CATalog:BASE:CORRection:IFEQualizer:SLOT<Slot>:TXFilter
value: List[str] = driver.catalog.correction.ifEqualizer.slot.txFilter.get(slot␣
↪= repcap.Slot.Default)
```

No command help available

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

return
filter_py: No help available

6.4 Cmwd

SCPI Commands :

```
INITiate:CMWD
STOP:CMWD
ABORt:CMWD
FETCh:CMWD
```

class CmwdCls

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

abort() → None

```
# SCPI: ABORt:CMWD
driver.cmwd.abort()
```

No command help available

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

```
# SCPI: ABORt:CMWD
driver.cmwd.abort_with_opc()
```

No command help available

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

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

fetch() → str

```
# SCPI: FETCh:CMWD
value: str = driver.cmwd.fetch()
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

return
result_string: No help available

initiate() → None

```
# SCPI: INITiate:CMWD
driver.cmwd.initiate()
```

No command help available

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

```
# SCPI: INITiate:CMWD
driver.cmwd.initiate_with_opc()
```

No command help available

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

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

stop() → None

```
# SCPI: STOP:CMWD
driver.cmwd.stop()
```

No command help available

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

```
# SCPI: STOP:CMWD
driver.cmwd.stop_with_opc()
```

No command help available

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

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

Cloning the Group

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

Subgroups

6.4.1 State

SCPI Command :

```
FETCh:CMWD:STATe
```

class StateCls

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

fetch() → ResourceState

```
# SCPI: FETCh:CMWD:STATe
value: enums.ResourceState = driver.cmwd.state.fetch()
```

No command help available

return
meas_state: No help available

6.5 Configure

SCPI Command :

```
CONFigure:BASE:FCONtrol
```

class ConfigureCls

Configure commands group definition. 54 total commands, 12 Subgroups, 1 group commands

get_fcontrol() → FanMode

```
# SCPI: CONFigure:BASE:FCONtrol
value: enums.FanMode = driver.configure.get_fcontrol()
```

Selects a fan control mode.

return
mode: LOW | NORMal | HIGH LOW: less cooling than in normal mode NORMal: default mode HIGH: more cooling than in normal mode

set_fcontrol(*mode: FanMode*) → None

```
# SCPI: CONFigure:BASE:FCONtrol
driver.configure.set_fcontrol(mode = enums.FanMode.HIGH)
```

Selects a fan control mode.

param mode
LOW | NORMal | HIGH LOW: less cooling than in normal mode NORMal: default mode HIGH: more cooling than in normal mode

Cloning the Group

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

Subgroups

6.5.1 Adjustment

SCPI Commands :

```
CONFigure:BASE:ADJustment:TYPE
CONFigure:BASE:ADJustment:VALue
CONFigure:BASE:ADJustment:SAVE
```

class AdjustmentCls

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

get_type_py() → OscillatorType

```
# SCPI: CONFigure:BASE:ADJustment:TYPE
value: enums.OscillatorType = driver.configure.adjustment.get_type_py()
```

No command help available

return
adj_type: No help available

get_value() → float

```
# SCPI: CONFigure:BASE:ADJustment:VALue
value: float = driver.configure.adjustment.get_value()
```

No command help available

return
adj_value: No help available

save() → None

```
# SCPI: CONFigure:BASE:ADJustment:SAVE
driver.configure.adjustment.save()
```

No command help available

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

```
# SCPI: CONFigure:BASE:ADJustment:SAVE
driver.configure.adjustment.save_with_opc()
```

No command help available

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

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

set_value(*adj_value: float*) → None

```
# SCPI: CONFigure:BASE:ADJustment:VALue
driver.configure.adjustment.set_value(adj_value = 1.0)
```

No command help available

param adj_value
No help available

6.5.2 Cmwd

SCPI Command :

```
CONFigure:CMWD:TIMeout
```

class CmwdCls

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

get_timeout() → float

```
# SCPI: CONFigure:CMWD:TIMeout
value: float = driver.configure.cmwd.get_timeout()
```

No command help available

return
timeout: No help available

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:CMWD:TIMeout
driver.configure.cmwd.set_timeout(timeout = 1.0)
```

No command help available

param timeout
No help available

6.5.3 Correction

class CorrectionCls

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

Cloning the Group

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

Subgroups

6.5.3.1 IfEqualizer

class IfEqualizerCls

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

Cloning the Group

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

Subgroups

6.5.3.1.1 Slot<Slot>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.correction.ifEqualizer.slot.repcap_slot_get()
driver.configure.correction.ifEqualizer.slot.repcap_slot_set(repcap.Slot.Nr1)
```

class SlotCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.correction.ifEqualizer.slot.clone()
```

Subgroups

6.5.3.1.1.1 RxFilter

class RxFilterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.correction.ifEqualizer.slot.rxFilter.clone()
```

Subgroups

6.5.3.1.1.2 Select

SCPI Command :

```
CONFigure:BASE:CORRection:IFEQualizer:SLOT<Slot>:RXFilter:SELect
```

class SelectCls

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

get(*slot=Slot.Default*) → List[bool]

```
# SCPI: CONFigure:BASE:CORRection:IFEQualizer:SLOT<Slot>:RXFilter:SELect
value: List[bool] = driver.configure.correction.ifEqualizer.slot.rxFilter.
↪select.get(slot = repcap.Slot.Default)
```

No command help available

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

return
select: No help available

set(*select: List[bool]*, *slot=Slot.Default*) → None

```
# SCPI: CONFigure:BASE:CORRection:IFEQualizer:SLOT<Slot>:RXFilter:SELect
driver.configure.correction.ifEqualizer.slot.rxFilter.select.set(select = [True,
↪ False, True], slot = repcap.Slot.Default)
```

No command help available

param select
No help available

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

6.5.3.1.1.3 TxFilter

class TxFilterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.correction.ifEqualizer.slot.txFilter.clone()
```

Subgroups

6.5.3.1.1.4 Select

SCPI Command :

```
CONFigure:BASE:CORRection:IFEQualizer:SLOT<Slot>:TXFilter:SELect
```

class SelectCls

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

get(*slot=Slot.Default*) → List[bool]

```
# SCPI: CONFigure:BASE:CORRection:IFEQualizer:SLOT<Slot>:TXFilter:SELect
value: List[bool] = driver.configure.correction.ifEqualizer.slot.txFilter.
↪select.get(slot = repcap.Slot.Default)
```

No command help available

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

return
select: No help available

set(*select: List[bool]*, *slot=Slot.Default*) → None

```
# SCPI: CONFigure:BASE:CORRection:IFEQualizer:SLOT<Slot>:TXFilter:SELect
driver.configure.correction.ifEqualizer.slot.txFilter.select.set(select = [True,
↪ False, True], slot = repcap.Slot.Default)
```

No command help available

param select
No help available

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

6.5.4 FreqCorrection

SCPI Commands :

```
CONFigure:BASE:FDCorrection:SAV
CONFigure:BASE:FDCorrection:RCL
CONFigure:FDCorrection:DEACtivate
CONFigure:FDCorrection:DEACtivate:ALL
```

class FreqCorrectionCls

FreqCorrection commands group definition. 16 total commands, 3 Subgroups, 4 group commands

deactivate(*connector: str*, *direction: RxTxDirection = None*, *rf_converter: RfConverterInPath = None*) → None

```
# SCPI: CONFigure:FDCorrection:DEACtivate
driver.configure.freqCorrection.deactivate(connector = rawAbc, direction =␣
↪enums.RxTxDirection.RX, rf_converter = enums.RfConverterInPath.RF1)
```

Deactivates any correction tables for a specific RF connector or a specific connector / converter combination. For bidirectional connectors, the tables can be deactivated for both directions or for one direction.

param connector
Selects a single RF connector For possible connector values, see 'Values for RF path selection'.

param direction
RXTX | RX | TX Specifies the direction for which the tables are deactivated. RX means input and TX means output. For a pure output connector, RX is ignored. RXTX: both directions (for output connector same effect as TX) RX: input (not allowed for output connector) TX: output Default: RXTX

param rf_converter
RF1 | RF2 | RF3 | RF4 RX and TX module in the path (RFn = RXn, TXn)

deactivate_all(*direction: RxTxDirection = None*, *table_path: str = None*) → None

```
# SCPI: CONFigure:FDCorrection:DEACtivate:ALL
driver.configure.freqCorrection.deactivate_all(direction = enums.RxTxDirection.
↪RX, table_path = rawAbc)
```

Deactivates all correction tables for all RF connectors of a selected subinstrument. For bidirectional connectors, the tables can be deactivated for both directions or for one direction.

param direction
RXTX | RX | TX Specifies the direction for which the tables are deactivated. RX means input and TX means output. For a pure output connector, RX is ignored. RXTX: both directions (for output connector same effect as TX) RX: input (not allowed for output connector) TX: output Default: RXTX

param table_path
string Selects the subinstrument If omitted: subinstrument addressed by the remote channel. 'instn': subinstrument n+1

recall(*table_path: str = None*) → None

```
# SCPI: CONFigure:BASE:FDCorrection:RCL
driver.configure.freqCorrection.recall(table_path = rawAbc)
```

Loads all correction tables for a selected subinstrument from the system drive into the RAM. This action is performed automatically when the R&S CMW application software is started. However, you can use the command to retrieve the correction tables after the disk contents have been modified. Or you can use it to undo changes and fall back to the tables stored on the system drive.

param table_path
string Selects the subinstrument If omitted: subinstrument addressed by the remote channel. 'instn': subinstrument n+1

save(*table_path: str = None*) → None

```
# SCPI: CONFigure:BASE:FDCorrection:SAV
driver.configure.freqCorrection.save(table_path = rawAbc)
```

Saves the correction tables for a selected subinstrument from the RAM to the system drive. This action is performed automatically when the R&S CMW application software is closed, for example, by pressing the standby key. However, you can use the command to save your work manually after creating or configuring correction tables.

param table_path
string Selects the subinstrument If omitted: subinstrument addressed by the remote channel. 'instn': subinstrument n+1

Cloning the Group

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

Subgroups

6.5.4.1 Activate

SCPI Command :

```
CONFigure:FDCorrection:ACTivate
```

class ActivateCls

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

class GetStruct

Response structure. Fields:

- Table_Rx: str: No parameter help available
- Table_Tx: str: No parameter help available

get(*connector: str*) → GetStruct

```
# SCPI: CONFigure:FDCorrection:ACTivate
value: GetStruct = driver.configure.freqCorrection.activate.get(connector =␣
↪rawAbc)
```

Activates a correction table for one or more signal paths using a specific RF connector. For bidirectional connectors, the table can be applied to both directions or to one direction. It is possible to assign different tables to the directions of a bidirectional connector. A table can be assigned to all paths using the connector or to paths with a specific connector / converter combination.

param connector
Selects a single RF connector For possible connector values, see 'Values for RF path selection'.

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

set(*connector: str*, *table: str*, *direction: RxTxDirection = None*, *rf_converter: RfConverterInPath = None*) → None

```
# SCPI: CONFigure:FDCorrection:ACTivate
driver.configure.freqCorrection.activate.set(connector = rawAbc, table = 'abc',␣
↪direction = enums.RxTxDirection.RX, rf_converter = enums.RfConverterInPath.
↪RF1)
```

Activates a correction table for one or more signal paths using a specific RF connector. For bidirectional connectors, the table can be applied to both directions or to one direction. It is possible to assign different tables to the directions of a bidirectional connector. A table can be assigned to all paths using the connector or to paths with a specific connector / converter combination.

param connector
Selects a single RF connector For possible connector values, see 'Values for RF path selection'.

param table
string To display a list of existing tables, use the command CONFigure:BASE:FDCorrection:CTABle:CATalog?. You can add the prefix 'instn/' to address subinstrument number n+1.

param direction
RXTX | RX | TX Specifies the direction to which the correction table is applied. RX means input and TX means output. For a pure output connector, RX is ignored. RXTX: both directions (for output connector same effect as TX) RX: input (not allowed for output connector) TX: output Default: RXTX

param rf_converter
RF1 | RF2 | RF3 | RF4 RX or TX module in the path (RFn = RXn / TXn) If omitted, the table is activated for any paths using the specified connector, independent of the used RX/TX module.

6.5.4.2 CorrectionTable

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:DELete
```

class CorrectionTableCls

CorrectionTable commands group definition. 10 total commands, 9 Subgroups, 1 group commands

delete(*table_name: str*) → None

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:DELete
driver.configure.freqCorrection.correctionTable.delete(table_name = 'abc')
```

Deletes a correction table from the RAM and the system drive.

param table_name
string To display a list of existing tables, use the command CONFigure:BASE:FDCorrection:CTABle:CATalog?. You can add the prefix 'instn/' to address subinstrument number n+1.

Cloning the Group

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

Subgroups

6.5.4.2.1 Add

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:ADD
```

class AddCls

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

set(*table_name: str*, *frequency: List[float] = None*, *correction: List[float] = None*) → None

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:ADD
driver.configure.freqCorrection.correctionTable.add.set(table_name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3], correction = [1.1, 2.2, 3.3])
```

Adds entries to an existing correction table. At least one parameter pair has to be specified. A command with an incomplete pair (e.g. <Frequency> without <Correction>) is ignored completely. You can add parameter pairs in any order. The table entries (pairs) are automatically sorted from lowest to highest frequency. For the supported frequency range, see 'Frequency range'.

param table_name
string To display a list of existing tables, use the command CONFigure:BASE:FDCorrection:CTABle:CATalog?. You can add the prefix 'instn/' to address subinstrument number n+1.

param frequency
numeric Unit: Hz

param correction
numeric Range: -50 dB to 90 dB, Unit: dB

6.5.4.2.2 Catalog

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:CATalog
```

class CatalogCls

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

get(*table_path: str = None*) → List[str]

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:CATalog
value: List[str] = driver.configure.freqCorrection.correctionTable.catalog.
↪get(table_path = rawAbc)
```

Returns the names of the correction tables currently stored on the system drive for a selected subinstrument.

param table_path
string Selects the subinstrument If omitted: subinstrument addressed by the remote channel. 'instn': subinstrument n+1

return
table_name: string Comma-separated list of table names as strings

6.5.4.2.3 Count

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:COUNt
```

class CountCls

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

get(*table_path: str = None*) → int

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:COUNt
value: int = driver.configure.freqCorrection.correctionTable.count.get(table_
↪path = rawAbc)
```

Returns the number of correction tables currently stored on the system drive for a selected subinstrument.

param table_path
string Selects the subinstrument If omitted: subinstrument addressed by the remote channel. 'instn': subinstrument n+1

return
table_count: decimal Number of tables

6.5.4.2.4 Create

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:CREate
```

class CreateCls

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

set(*table_name: str*, *frequency: List[float] = None*, *correction: List[float] = None*) → None

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:CREate
driver.configure.freqCorrection.correctionTable.create.set(table_name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3], correction = [1.1, 2.2, 3.3])
```

Creates a correction table for frequency-dependent attenuation and stores it in the RAM. If a table with the given name exists for the addressed subinstrument, it is overwritten. The parameter pairs <Frequency>, <Correction> are used to fill the table. A command with an incomplete pair (e.g. <Frequency> without <Correction>) is ignored completely. To add entries to an existing table, see method RsCmwBase.Configure.FreqCorrection.CorrectionTable.Add.set. You can enter parameter pairs in any order. The table entries (pairs) are automatically sorted from lowest to highest frequency. For the supported frequency range, see 'Frequency range'.

param table_name
string The table name is used to identify the table in other commands and to store

the table on the system drive. The string must comply to Windows™ file name conventions, see 'Mass memory commands'. You can add the prefix 'instn/' to address subinstrument number n+1. Example: 'inst2/mytable' means 'mytable' for subinstrument number 3.

param frequency
numeric Unit: Hz

param correction
numeric Range: -50 dB to 90 dB, Unit: dB

6.5.4.2.5 DeleteAll

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:DELete:ALL
```

class DeleteAllCls

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

set(*table_path: str = None*) → None

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:DELete:ALL
driver.configure.freqCorrection.correctionTable.deleteAll.set(table_path =␣
↪rawAbc)
```

Deletes all correction tables for a selected subinstrument from the RAM and the system drive.

param table_path
string Selects the subinstrument If omitted: subinstrument addressed by the remote channel. 'instn': subinstrument n+1

6.5.4.2.6 Details

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:DETails
```

class DetailsCls

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

class GetStruct

Response structure. Fields:

- Frequency: List[float]: No parameter help available
- Correction: List[float]: No parameter help available

get(*table_name: str*, *start_index: float = None*, *count: float = None*) → GetStruct

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:DETails
value: GetStruct = driver.configure.freqCorrection.correctionTable.details.
↪get(table_name = 'abc', start_index = 1.0, count = 1.0)
```

Returns the entries of a correction table.

param table_name
string To display a list of existing tables, use the command CONFigure:BASE:FDCorrection:CTABle:CATalog?. You can add the prefix 'instn/' to address subinstrument number n+1.

param start_index
numeric Index number of the first entry to be listed. The first entry of a table has index number 0. Default: 0

param count
numeric Maximum number of entries to be listed. By default, all entries from StartIndex to the end of the table are listed.

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

6.5.4.2.7 Erase

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:ERASe
```

class EraseCls

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

set(*table_name: str*, *frequency: List[float] = None*) → None

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:ERASe
driver.configure.freqCorrection.correctionTable.erase.set(table_name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3])
```

Removes one or more selected entries from a correction table. Each table entry consists of a frequency value and a correction value. Entries to be removed are selected via their frequency values. For the supported frequency range, see 'Frequency range'.

param table_name
string To display a list of existing tables, use the command CONFigure:BASE:FDCorrection:CTABle:CATalog?. You can add the prefix 'instn/' to address subinstrument number n+1.

param frequency
numeric Selects the table entry to be removed. The value must match the frequency of an existing table entry. To remove several entries, specify a comma-separated list of frequencies. Unit: Hz

6.5.4.2.8 Exist

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:EXISt
```

class ExistCls

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

get(*table_name: str*) → int

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:EXISt
value: int = driver.configure.freqCorrection.correctionTable.exist.get(table_
↪name = 'abc')
```

Queries whether a correction table with the specified name exists or not.

param table_name
string You can add the prefix 'instn/' to the table name, to address subinstrument number n+1.

return
exists: 0 | 1 0: table does not exist 1: table exists

6.5.4.2.9 Length

SCPI Command :

```
CONFigure:BASE:FDCorrection:CTABle:LENGth
```

class LengthCls

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

get(*table_name: str*) → int

```
# SCPI: CONFigure:BASE:FDCorrection:CTABle:LENGth
value: int = driver.configure.freqCorrection.correctionTable.length.get(table_
↪name = 'abc')
```

Returns the number of entries (i.e. pairs of frequency and attenuation) of a correction table.

param table_name
string To display a list of existing tables, use the command method **RsCmwBase.Configure.FreqCorrection.CorrectionTable.Catalog.get_**. You can add the prefix 'instn/' to address subinstrument number n+1.

return
table_length: decimal Number of table entries

6.5.4.3 Usage

SCPI Command :

```
CONFigure:FDCorrection:USAGe
```

class UsageCls

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

class GetStruct

Response structure. Fields:

- Not_Avail_Rx: str: No parameter help available
- Correction_Table_Rx: str: No parameter help available
- Not_Avail_Tx: str: No parameter help available
- Correction_Table_Tx: str: No parameter help available

get(*connector: str*, *rf_converter: RfConverterInPath = None*) → GetStruct

```
# SCPI: CONFigure:FDCorrection:USAGe
value: GetStruct = driver.configure.freqCorrection.usage.get(connector = rawAbc,
↪ rf_converter = enums.RfConverterInPath.RF1)
```

Lists the correction tables assigned to a specific RF connector or a specific connector / converter combination.

param connector
Selects a single RF connector For possible connector values, see 'Values for RF path selection'.

param rf_converter
RF1 | RF2 | RF3 | RF4 RX and TX module in the path (RFn = RXn, TXn) If the specified converter value is incompatible with the connector or the results are ambiguous because this parameter is omitted, NAV is returned.

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

6.5.5 Ipcr

SCPI Commands :

```
CONFigure:BASE:IPCR:ENABle
CONFigure:BASE:IPCR:IDENt
```

class IpcrCls

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

get_enable() → List[bool]

```
# SCPI: CONFigure:BASE:IPCR:ENABle
value: List[bool] = driver.configure.ipcr.get_enable()
```

No command help available

return
enable: No help available

get_ident() → List[str]

```
# SCPI: CONFigure:BASE:IPCR:IDENt
value: List[str] = driver.configure.ipcr.get_ident()
```

No command help available

return
ident: No help available

set_enable(*enable: List[bool]*) → None

```
# SCPI: CONFigure:BASE:IPCR:ENABle
driver.configure.ipcr.set_enable(enable = [True, False, True])
```

No command help available

param enable
No help available

6.5.6 IpSet

class IpSetCls

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

Cloning the Group

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

Subgroups

6.5.6.1 NwAdapter<NwAdapter>

RepCap Settings

```
# Range: Adapter1 .. Adapter5
rc = driver.configure.ipSet.nwAdapter.repcap_nwAdapter_get()
driver.configure.ipSet.nwAdapter.repcap_nwAdapter_set(repcap.NwAdapter.Adapter1)
```

SCPI Command :

```
CONFigure:BASE:IPSet:NWADapter<n>
```

class NwAdapterCls

NwAdapter commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: NwAdapter, default value after init: NwAdapter.Adapter1

class GetStruct

Response structure. Fields:

- Nw_Adapter_Name: str: string Name of the network adapter, e.g. 'LAN Remote' for n = 1 A returned OFF indicates that the selected value n is not assigned to a network adapter.
- Set_Subnet_Conform: bool: ON | OFF | 1 | 0 To assign a subnet conform IP address, set 1 or ON. To try again, set first 0 or OFF, then again 1 or ON. A query returns whether the last set value was 0 or 1.
- Ip_Address: str: string IP address (to be) assigned, see Status.
- Status: enums.AdjustStatus: NADJust | ADJust State indicating whether the returned IP address has been successfully assigned to the network adapter (ADJust) or not (NADJust) .

get(*nwAdapter=NwAdapter.Default*) → GetStruct

```
# SCPI: CONFigure:BASE:IPSet:NWADapter<n>
value: GetStruct = driver.configure.ipSet.nwAdapter.get(nwAdapter = repcap.
↪NwAdapter.Default)
```

Assigns a subnet conform IP address to a network adapter of the instrument, selected via index <n> or returns information about this network adapter. A query returns <NWAdapterName>, <SetSubnetConform>, <IPAddress>, <Status>.

param nwAdapter
optional repeated capability selector. Default value: Adapter1 (settable in the interface 'NwAdapter')

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

set(*set_subnet_conform: bool*, *nwAdapter=NwAdapter.Default*) → None

```
# SCPI: CONFigure:BASE:IPSet:NWADapter<n>
driver.configure.ipSet.nwAdapter.set(set_subnet_conform = False, nwAdapter =␣
↪repcap.NwAdapter.Default)
```

Assigns a subnet conform IP address to a network adapter of the instrument, selected via index <n> or returns information about this network adapter. A query returns <NWAdapterName>, <SetSubnetConform>, <IPAddress>, <Status>.

param set_subnet_conform
ON | OFF | 1 | 0 To assign a subnet conform IP address, set 1 or ON. To try again, set first 0 or OFF, then again 1 or ON. A query returns whether the last set value was 0 or 1.

param nwAdapter
optional repeated capability selector. Default value: Adapter1 (settable in the interface 'NwAdapter')

Cloning the Group

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

6.5.7 Mmonitor

class MmonitorCls

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

Cloning the Group

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

Subgroups

6.5.7.1 IpAddress<IpAddress>

RepCap Settings

```
# Range: Addr1 .. Addr3
rc = driver.configure.mmonitor.ipAddress.repcap_ipAddress_get()
driver.configure.mmonitor.ipAddress.repcap_ipAddress_set(repcap.IpAddress.Addr1)
```

SCPI Command :

```
CONFigure:BASE:MMONitor:IPADdress<n>
```

class IpAddressCls

IpAddress commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: IpAddress, default value after init: IpAddress.Addr1

class IpAddressStruct

Response structure. Fields:

- First_Segment: int: numeric First octet of the IP address, not configurable Range: 0 to 255
- Second_Segment: int: numeric Second octet of the IP address, not configurable Range: 0 to 255
- System_Id: int: numeric Third octet of the IP address Range: 5 to 255
- Local_Id: int: numeric Fourth octet of the IP address Range: 1 to 254

get(*ipAddress=IpAddress.Default*) → IpAddressStruct

```
# SCPI: CONFigure:BASE:MMONitor:IPADdress<n>
value: IpAddressStruct = driver.configure.mmonitor.ipAddress.get(ipAddress =␣
↪repcap.IpAddress.Default)
```

Configures the IP address pool for logging of signaling messages via an external PC. The pool contains three IP addresses of external logging PCs. The first two octets cannot be configured. For a setting command, you can specify any values within the allowed range - they are ignored. A query returns the active values resulting from the subnet configuration, see CONFigure:BASE:IPSet:SNODe.

param ipAddress
optional repeated capability selector. Default value: Addr1 (settable in the interface 'IpAddress')

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

set(*first_segment: int*, *second_segment: int*, *system_id: int*, *local_id: int*, *ipAddress=IpAddress.Default*) → None

```
# SCPI: CONFigure:BASE:MMONitor:IPADdress<n>
driver.configure.mmonitor.ipAddress.set(first_segment = 1, second_segment = 1,␣
↪system_id = 1, local_id = 1, ipAddress = repcap.IpAddress.Default)
```

Configures the IP address pool for logging of signaling messages via an external PC. The pool contains three IP addresses of external logging PCs. The first two octets cannot be configured. For a setting command, you can specify any values within the allowed range - they are ignored. A query returns the active values resulting from the subnet configuration, see CONFigure:BASE:IPSet:SNODe.

param first_segment
numeric First octet of the IP address, not configurable Range: 0 to 255

param second_segment
numeric Second octet of the IP address, not configurable Range: 0 to 255

param system_id
numeric Third octet of the IP address Range: 5 to 255

param local_id
numeric Fourth octet of the IP address Range: 1 to 254

param ipAddress
optional repeated capability selector. Default value: Addr1 (settable in the interface 'IpAddress')

Cloning the Group

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

6.5.8 MultiCmw

SCPI Command :

```
CONFigure:BASE:MCMW:REARrange
```

class MultiCmwCls

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

set_rearrange(*box_nr: List[BoxNumber]*) → None

```
# SCPI: CONFigure:BASE:MCMW:REARrange
driver.configure.multiCmw.set_rearrange(box_nr = [BoxNumber.BOX1, BoxNumber.
↪NAV])
```

No command help available

param box_nr
No help available

Cloning the Group

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

Subgroups

6.5.8.1 Identify

SCPI Command :

```
CONFigure:BASE:MCMW:IDENtify:BTIMe
```

class IdentifyCls

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

get_btime() → int

```
# SCPI: CONFigure:BASE:MCMW:IDENtify:BTIMe
value: int = driver.configure.multiCmw.identify.get_btime()
```

No command help available

return
blinking_time: No help available

set_btime(*blinking_time: int*) → None

```
# SCPI: CONFigure:BASE:MCMW:IDENtify:BTIMe
driver.configure.multiCmw.identify.set_btime(blinking_time = 1)
```

No command help available

param blinking_time
No help available

6.5.9 Mutex

SCPI Commands :

```
CONFigure:MUTex:UNLock
CONFigure:MUTex:UNDefine
CONFigure:MUTex:CATalog
```

class MutexCls

Mutex commands group definition. 6 total commands, 3 Subgroups, 3 group commands

class CatalogStruct

Structure for reading output parameters. Fields:

- Name: str: No parameter help available
- Def_Timeout: int: No parameter help available
- State: enums.MutexState: No parameter help available

get_catalog() → CatalogStruct

```
# SCPI: CONFigure:MUTex:CATalog
value: CatalogStruct = driver.configure.mutex.get_catalog()
```

No command help available

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

set_undefine(*name: str*) → None

```
# SCPI: CONFigure:MUTex:UNDefine
driver.configure.mutex.set_undefine(name = 'abc')
```

No command help available

param name
No help available

unlock(*name: str*, *key: float*) → None

```
# SCPI: CONFigure:MUTex:UNLock
driver.configure.mutex.unlock(name = 'abc', key = 1.0)
```

No command help available

param name
No help available

param key
No help available

Cloning the Group

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

Subgroups

6.5.9.1 Define

SCPI Command :

```
CONFigure:MUTex:DEFine
```

class DefineCls

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

set(*name: str*, *timeout: float*, *scope: ValidityScopeA = None*) → None

```
# SCPI: CONFigure:MUTex:DEFine
driver.configure.mutex.define.set(name = 'abc', timeout = 1.0, scope = enums.
↪ValidityScopeA.GLOBal)
```

No command help available

param name
No help available

param timeout
No help available

param scope
No help available

6.5.9.2 Lock

SCPI Command :

```
CONFigure:MUTex:LOCK
```

class LockCls

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

get(*name: str*, *timeout: float = None*) → int

```
# SCPI: CONFigure:MUTex:LOCK
value: int = driver.configure.mutex.lock.get(name = 'abc', timeout = 1.0)
```

No command help available

param name
No help available

param timeout
No help available

return
key: No help available

6.5.9.3 State

SCPI Command :

```
CONFigure:MUTex:STATe
```

class StateCls

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

class GetStruct

Response structure. Fields:

- State: enums.MutexState: No parameter help available
- Key: int: No parameter help available

get(*name: str*, *action: MutexAction = None*, *timeout: float = None*) → GetStruct

```
# SCPI: CONFigure:MUTex:STATe
value: GetStruct = driver.configure.mutex.state.get(name = 'abc', action =␣
↪enums.MutexAction.DONothing, timeout = 1.0)
```

No command help available

param name
No help available

param action
No help available

param timeout
No help available

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

6.5.10 Semaphore

SCPI Commands :

```
CONFigure:SEMaphore:CATalog
CONFigure:SEMaphore:UNDefine
```

class SemaphoreCls

Semaphore commands group definition. 6 total commands, 4 Subgroups, 2 group commands

class CatalogStruct

Structure for reading output parameters. Fields:

- Name: str: No parameter help available
- Def_Timeout: int: No parameter help available
- Def_Count: int: No parameter help available

- Scope: enums.ValidityScopeA: No parameter help available

get_catalog() → CatalogStruct

```
# SCPI: CONFigure:SEMaphore:CATalog
value: CatalogStruct = driver.configure.semaphore.get_catalog()
```

No command help available

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

set_undefine(*name: str*) → None

```
# SCPI: CONFigure:SEMaphore:UNDefine
driver.configure.semaphore.set_undefine(name = 'abc')
```

No command help available

param name
No help available

Cloning the Group

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

Subgroups

6.5.10.1 Acquire

SCPI Command :

```
CONFigure:SEMaphore:ACQuire
```

class AcquireCls

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

get(*name: str*) → int

```
# SCPI: CONFigure:SEMaphore:ACQuire
value: int = driver.configure.semaphore.acquire.get(name = 'abc')
```

No command help available

param name
No help available

return
key: No help available

6.5.10.2 Count

SCPI Command :

```
CONFigure:SEMaphore:COUNt
```

class CountCls

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

get(*name: str*) → int

```
# SCPI: CONFigure:SEMaphore:COUNt
value: int = driver.configure.semaphore.count.get(name = 'abc')
```

No command help available

param name
No help available

return
count: No help available

6.5.10.3 Define

SCPI Command :

```
CONFigure:SEMaphore:DEFine
```

class DefineCls

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

set(*name: str*, *def_timeout: float*, *def_count: int*, *scope: ValidityScopeA = None*) → None

```
# SCPI: CONFigure:SEMaphore:DEFine
driver.configure.semaphore.define.set(name = 'abc', def_timeout = 1.0, def_
↪count = 1, scope = enums.ValidityScopeA.GLOBal)
```

No command help available

param name
No help available

param def_timeout
No help available

param def_count
No help available

param scope
No help available

6.5.10.4 Release

SCPI Command :

```
CONFigure:SEMaphore:RELease
```

class ReleaseCls

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

set(*name: str*, *key: int*) → None

```
# SCPI: CONFigure:SEMaphore:RELease
driver.configure.semaphore.release.set(name = 'abc', key = 1)
```

No command help available

param name
No help available

param key
No help available

6.5.11 SingleCmw

class SingleCmwCls

SingleCmw commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.5.11.1 FreqCorrection

SCPI Command :

```
CONFigure:CMWS:FDCorrection:DEACtivate:ALL
```

class FreqCorrectionCls

FreqCorrection commands group definition. 8 total commands, 3 Subgroups, 1 group commands

deactivate_all(*table_path: str = None*) → None

```
# SCPI: CONFigure:CMWS:FDCorrection:DEACtivate:ALL
driver.configure.singleCmw.freqCorrection.deactivate_all(table_path = rawAbc)
```

No command help available

param table_path
No help available

Cloning the Group

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

Subgroups

6.5.11.1.1 Activate

class ActivateCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.singleCmw.freqCorrection.activate.clone()
```

Subgroups

6.5.11.1.1.1 Rx

SCPI Command :

```
CONFigure:CMWS:FDCorrection:ACTivate:RX
```

class RxCls

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

class SetStruct

Structure for setting input parameters. Contains optional setting parameters. Fields:

- Connector_Bench: str: No parameter help available
- Table_1: str: No parameter help available
- Table_2: str: No parameter help available
- Table_3: str: No parameter help available
- Table_4: str: No parameter help available
- Table_5: str: No parameter help available
- Table_6: str: No parameter help available
- Table_7: str: No parameter help available
- Table_8: str: No parameter help available

set(*structure: SetStruct*) → None

```
# SCPI: CONFigure:CMWS:FDCorrection:ACTivate:RX
structure = driver.configure.singleCmw.freqCorrection.activate.rx.SetStruct()
structure.Connector_Bench: str = rawAbc
structure.Table_1: str = 'abc'
structure.Table_2: str = 'abc'
structure.Table_3: str = 'abc'
structure.Table_4: str = 'abc'
structure.Table_5: str = 'abc'
structure.Table_6: str = 'abc'
structure.Table_7: str = 'abc'
structure.Table_8: str = 'abc'
driver.configure.singleCmw.freqCorrection.activate.rx.set(structure)
```

No command help available

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

6.5.11.1.1.2 Tx

SCPI Command :

```
CONFigure:CMWS:FDCorrection:ACTivate:TX
```

class TxCls

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

class SetStruct

Structure for setting input parameters. Contains optional setting parameters. Fields:

- Connector_Bench: str: No parameter help available
- Table_1: str: No parameter help available
- Table_2: str: No parameter help available
- Table_3: str: No parameter help available
- Table_4: str: No parameter help available
- Table_5: str: No parameter help available
- Table_6: str: No parameter help available
- Table_7: str: No parameter help available
- Table_8: str: No parameter help available

set(*structure: SetStruct*) → None

```
# SCPI: CONFigure:CMWS:FDCorrection:ACTivate:TX
structure = driver.configure.singleCmw.freqCorrection.activate.tx.SetStruct()
structure.Connector_Bench: str = rawAbc
structure.Table_1: str = 'abc'
structure.Table_2: str = 'abc'
structure.Table_3: str = 'abc'
structure.Table_4: str = 'abc'
```

(continues on next page)

(continued from previous page)

```
structure.Table_5: str = 'abc'
structure.Table_6: str = 'abc'
structure.Table_7: str = 'abc'
structure.Table_8: str = 'abc'
driver.configure.singleCmw.freqCorrection.activate.tx.set(structure)
```

No command help available

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

6.5.11.1.2 Deactivate

class DeactivateCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.singleCmw.freqCorrection.deactivate.clone()
```

Subgroups

6.5.11.1.2.1 Rx

SCPI Command :

```
CONFigure:CMWS:FDCorrection:DEACtivate:RX
```

class RxCls

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

set_value(*connector_bench: str*) → None

```
# SCPI: CONFigure:CMWS:FDCorrection:DEACtivate:RX
driver.configure.singleCmw.freqCorrection.deactivate.rx.set_value(connector_
↪bench = rawAbc)
```

No command help available

param connector_bench
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.singleCmw.freqCorrection.deactivate.rx.clone()
```

Subgroups

6.5.11.1.2.2 All

SCPI Command :

```
CONFigure:CMWS:FDCorrection:DEACtivate:RX:ALL
```

class AllCls

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

set(*table_path: str = None*) → None

```
# SCPI: CONFigure:CMWS:FDCorrection:DEACtivate:RX:ALL
driver.configure.singleCmw.freqCorrection.deactivate.rx.all.set(table_path =␣
↪rawAbc)
```

No command help available

param table_path
No help available

6.5.11.1.2.3 Tx

SCPI Command :

```
CONFigure:CMWS:FDCorrection:DEACtivate:TX
```

class TxCls

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

set_value(*connector_bench: str*) → None

```
# SCPI: CONFigure:CMWS:FDCorrection:DEACtivate:TX
driver.configure.singleCmw.freqCorrection.deactivate.tx.set_value(connector_
↪bench = rawAbc)
```

No command help available

param connector_bench
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.singleCmw.freqCorrection.deactivate.tx.clone()
```

Subgroups

6.5.11.1.2.4 All

SCPI Command :

```
CONFigure:CMWS:FDCorrection:DEACtivate:TX:ALL
```

class AllCls

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

set(*table_path: str = None*) → None

```
# SCPI: CONFigure:CMWS:FDCorrection:DEACtivate:TX:ALL
driver.configure.singleCmw.freqCorrection.deactivate.tx.all.set(table_path =␣
↪rawAbc)
```

No command help available

param table_path
No help available

6.5.11.1.3 Usage

SCPI Command :

```
CONFigure:CMWS:FDCorrection:USAGe
```

class UsageCls

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

class GetStruct

Response structure. Fields:

- Correction_Table_Rx: str: No parameter help available
- Correction_Table_Tx: str: No parameter help available

get(*connector: str*) → GetStruct

```
# SCPI: CONFigure:CMWS:FDCorrection:USAGe
value: GetStruct = driver.configure.singleCmw.freqCorrection.usage.
↪get(connector = rawAbc)
```

No command help available

param connector
No help available

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

6.5.12 Spoint

SCPI Commands :

```
CONFigure:SPOint:CATalog
CONFigure:SPOint:UNDefine
```

class SpointCls

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

class CatalogStruct

Structure for reading output parameters. Fields:

- Name: str: No parameter help available
- Def_Timeout: int: No parameter help available
- Def_Count: int: No parameter help available
- Scope: enums.ValidityScopeA: No parameter help available

get_catalog() → CatalogStruct

```
# SCPI: CONFigure:SPOint:CATalog
value: CatalogStruct = driver.configure.spoint.get_catalog()
```

No command help available

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

set_undefine(*name: str*) → None

```
# SCPI: CONFigure:SPOint:UNDefine
driver.configure.spoint.set_undefine(name = 'abc')
```

No command help available

param name
No help available

Cloning the Group

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

Subgroups

6.5.12.1 Define

SCPI Command :

```
CONFigure:SPOint:DEFine
```

class DefineCls

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

set(*name: str*, *def_timeout: float*, *def_count: int*, *scope: ValidityScopeA = None*) → None

```
# SCPI: CONFigure:SPOint:DEFine
driver.configure.spoint.define.set(name = 'abc', def_timeout = 1.0, def_count =␣
↪1, scope = enums.ValidityScopeA.GLOBal)
```

No command help available

param name
No help available

param def_timeout
No help available

param def_count
No help available

param scope
No help available

6.5.12.2 Join

SCPI Command :

```
CONFigure:SPOint:JOIN
```

class JoinCls

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

class GetStruct

Response structure. Fields:

- Count: int: No parameter help available
- Result: enums.SyncResult: No parameter help available

get(*name: str*, *action: JoinAction = None*, *polling: SyncPolling = None*, *poll_interval: float = None*) → GetStruct

```
# SCPI: CONFigure:SPOint:JOIN
value: GetStruct = driver.configure.spoint.join.get(name = 'abc', action =␣
↪enums.JoinAction.CTASk, polling = enums.SyncPolling.NPOLling, poll_interval =␣
↪1.0)
```

No command help available

param name
No help available

param action
No help available

param polling
No help available

param poll_interval
No help available

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

6.5.12.3 Rewait

SCPI Command :

```
CONFigure:SPOint:REWait
```

class RewaitCls

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

class GetStruct

Response structure. Fields:

- Count: int: No parameter help available
- Result: enums.SyncResult: No parameter help available

get(*name: str*) → GetStruct

```
# SCPI: CONFigure:SPOint:REWait
value: GetStruct = driver.configure.spoint.rewait.get(name = 'abc')
```

No command help available

param name
No help available

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

6.6 Correction

class CorrectionCls

Correction commands group definition. 13 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.1 IfEqualizer

SCPI Commands :

```
INITiate:BASE:CORRection:IFEQualizer
ABORt:BASE:CORRection:IFEQualizer
```

class IfEqualizerCls

IfEqualizer commands group definition. 13 total commands, 3 Subgroups, 2 group commands

abort() → None

```
# SCPI: ABORt:BASE:CORRection:IFEQualizer
driver.correction.ifEqualizer.abort()
```

No command help available

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

```
# SCPI: ABORt:BASE:CORRection:IFEQualizer
driver.correction.ifEqualizer.abort_with_opc()
```

No command help available

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

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

initiate() → None

```
# SCPI: INITiate:BASE:CORRection:IFEQualizer
driver.correction.ifEqualizer.initiate()
```

No command help available

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

```
# SCPI: INITiate:BASE:CORRection:IFEQualizer
driver.correction.ifEqualizer.initiate_with_opc()
```

No command help available

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

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

Cloning the Group

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

Subgroups

6.6.1.1 Slot<Slot>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.correction.ifEqualizer.slot.repcap_slot_get()
driver.correction.ifEqualizer.slot.repcap_slot_set(repcap.Slot.Nr1)
```

class SlotCls

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

Cloning the Group

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

Subgroups

6.6.1.1.1 RxFilter

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:SLOT<Slot>:RXFilter
```

class RxFilterCls

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

fetch(*slot=Slot.Default*) → List[CorrResult]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:SLOT<Slot>:RXFilter
value: List[enums.CorrResult] = driver.correction.ifEqualizer.slot.rxFilter.
↪fetch(slot = repcap.Slot.Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

return
value: No help available

6.6.1.1.2 TxFilter

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:SLOT<Slot>:TXFilter
```

class TxFilterCls

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

fetch(*slot=Slot.Default*) → List[CorrResult]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:SLOT<Slot>:TXFilter
value: List[enums.CorrResult] = driver.correction.ifEqualizer.slot.txFilter.
↪fetch(slot = repcap.Slot.Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

return
value: No help available

6.6.1.2 State

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:STATe
```

class StateCls

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

fetch() → ResourceState

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:STATe
value: enums.ResourceState = driver.correction.ifEqualizer.state.fetch()
```

No command help available

return
meas_status: No help available

6.6.1.3 Trace

class TraceCls

Trace commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.1.3.1 Gdelay

class GdelayCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.clone()
```

Subgroups

6.6.1.3.1.1 Corrected

class CorrectedCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.corrected.clone()
```

Subgroups

6.6.1.3.1.2 Slot<Slot>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.repcap_slot_get()
driver.correction.ifEqualizer.trace.gdelay.corrected.slot.repcap_slot_set(repcap.Slot.
↪Nr1)
```

class SlotCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.clone()
```

Subgroups

6.6.1.3.1.3 RxFilter<RxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.rxFilter.repcap_rxFilter_
↪get()
driver.correction.ifEqualizer.trace.gdelay.corrected.slot.rxFilter.repcap_rxFilter_
↪set(repcap.RxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:CORRected:SLOT<Slot>:RXFilter<Filter>
```

class RxFilterCls

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

fetch(*slot=Slot.Default*, *rxFilter=RxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:CORRected:SLOT<Slot>
↪:RXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.
↪rxFilter.fetch(slot = repcap.Slot.Default, rxFilter = repcap.RxFilter.Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.rxFilter.clone()
```

6.6.1.3.1.4 TxFilter<TxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.txFilter.repcap_txFilter_
↪get()
driver.correction.ifEqualizer.trace.gdelay.corrected.slot.txFilter.repcap_txFilter_
↪set(repcap.TxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:CORRected:SLOT<Slot>:TXFilter<Filter>
```

class TxFilterCls

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

fetch(*slot=Slot.Default*, *txFilter=TxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:CORRected:SLOT<Slot>
↪:TXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.
↪txFilter.fetch(slot = repcap.Slot.Default, txFilter = repcap.TxFilter.Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.corrected.slot.txFilter.clone()
```

6.6.1.3.1.5 Uncorrected

class UncorrectedCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.uncorrected.clone()
```

Subgroups

6.6.1.3.1.6 Slot<Slot>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.repcap_slot_get()
driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.repcap_slot_set(repcap.Slot.
↪Nr1)
```

class SlotCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.clone()
```

Subgroups

6.6.1.3.1.7 RxFilter<RxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.rxFilter.repcap_
↪rxFilter_get()
driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.rxFilter.repcap_rxFilter_
↪set(repcap.RxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:UNCorrected:SLOT<Slot>:RXFilter<Filter>
```

class RxFilterCls

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

fetch(*slot=Slot.Default*, *rxFilter=RxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:UNCorrected:SLOT<Slot>
↪:RXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.gdelay.uncorrected.
↪slot.rxFilter.fetch(slot = repcap.Slot.Default, rxFilter = repcap.RxFilter.
↪Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.rxFilter.clone()
```

6.6.1.3.1.8 TxFilter<TxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.txFilter.repcap_
↪txFilter_get()
driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.txFilter.repcap_txFilter_
↪set(repcap.TxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:UNCorrected:SLOT<Slot>:TXFilter<Filter>
```

class TxFilterCls

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

fetch(*slot=Slot.Default*, *txFilter=TxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:GDELay:UNCorrected:SLOT<Slot>
↪:TXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.gdelay.uncorrected.
↪slot.txFilter.fetch(slot = repcap.Slot.Default, txFilter = repcap.TxFilter.
↪Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.gdelay.uncorrected.slot.txFilter.clone()
```

6.6.1.3.2 Magnitude

class MagnitudeCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.clone()
```

Subgroups

6.6.1.3.2.1 Corrected

class CorrectedCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.corrected.clone()
```

Subgroups

6.6.1.3.2.2 Slot<Slot>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.correction.ifEqualizer.trace.magnitude.corrected.slot.repcap_slot_get()
driver.correction.ifEqualizer.trace.magnitude.corrected.slot.repcap_slot_set(repcap.Slot.
↪Nr1)
```

class SlotCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.corrected.slot.clone()
```

Subgroups

6.6.1.3.2.3 RxFilter<RxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.magnitude.corrected.slot.rxFilter.repcap_
↪rxFilter_get()
driver.correction.ifEqualizer.trace.magnitude.corrected.slot.rxFilter.repcap_rxFilter_
↪set(repcap.RxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:CORRected:SLOT<Slot>:RXFilter<Filter>
```

class RxFilterCls

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

fetch(*slot=Slot.Default*, *rxFilter=RxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:CORRected:SLOT<Slot>
↪:RXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.magnitude.corrected.
↪slot.rxFilter.fetch(slot = repcap.Slot.Default, rxFilter = repcap.RxFilter.
↪Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.corrected.slot.rxFilter.clone()
```

6.6.1.3.2.4 TxFilter<TxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.magnitude.corrected.slot.txFilter.repcap_
↪txFilter_get()
driver.correction.ifEqualizer.trace.magnitude.corrected.slot.txFilter.repcap_txFilter_
↪set(repcap.TxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:CORRected:SLOT<Slot>:TXFilter<Filter>
```

class TxFilterCls

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

fetch(*slot=Slot.Default*, *txFilter=TxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:CORRected:SLOT<Slot>
↪:TXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.magnitude.corrected.
↪slot.txFilter.fetch(slot = repcap.Slot.Default, txFilter = repcap.TxFilter.
↪Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.corrected.slot.txFilter.clone()
```

6.6.1.3.2.5 Uncorrected

class UncorrectedCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.uncorrected.clone()
```

Subgroups

6.6.1.3.2.6 Slot<Slot>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.repcap_slot_get()
driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.repcap_slot_set(repcap.
↪Slot.Nr1)
```

class SlotCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.clone()
```

Subgroups

6.6.1.3.2.7 RxFilter<RxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.rxFilter.repcap_
↪rxFilter_get()
driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.rxFilter.repcap_rxFilter_
↪set(repcap.RxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:UNCorrected:SLOT<Slot>:RXFilter<Filter>
```

class RxFilterCls

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

fetch(*slot=Slot.Default*, *rxFilter=RxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:UNCorrected:SLOT<Slot>
↪:RXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.magnitude.uncorrected.
↪slot.rxFilter.fetch(slot = repcap.Slot.Default, rxFilter = repcap.RxFilter.
↪Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.rxFilter.clone()
```

6.6.1.3.2.8 TxFilter<TxFilter>

RepCap Settings

```
# Range: Nr1 .. Nr10
rc = driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.txFilter.repcap_
↪txFilter_get()
driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.txFilter.repcap_txFilter_
↪set(repcap.TxFilter.Nr1)
```

SCPI Command :

```
FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:UNCorrected:SLOT<Slot>:TXFilter<Filter>
```

class TxFilterCls

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

fetch(*slot=Slot.Default*, *txFilter=TxFilter.Default*) → List[float]

```
# SCPI: FETCh:BASE:CORRection:IFEQualizer:TRACe:MAGNitude:UNCorrected:SLOT<Slot>
↪:TXFilter<Filter>
value: List[float] = driver.correction.ifEqualizer.trace.magnitude.uncorrected.
↪slot.txFilter.fetch(slot = repcap.Slot.Default, txFilter = repcap.TxFilter.
↪Default)
```

No command help available

Use RsCmwBase.reliability.last_value to read the updated reliability indicator.

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

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

return
value: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.correction.ifEqualizer.trace.magnitude.uncorrected.slot.txFilter.clone()
```

6.7 Diagnostic

SCPI Command :

```
DIAGnostic:SDBM
```

class DiagnosticCls

Diagnostic commands group definition. 68 total commands, 17 Subgroups, 1 group commands

set_sdbm(*text: str*) → None

```
# SCPI: DIAGnostic:SDBM
driver.diagnostic.set_sdbm(text = 'abc')
```

No command help available

param text
No help available

Cloning the Group

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

Subgroups

6.7.1 Access

class AccessCls

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

Cloning the Group

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

Subgroups

6.7.1.1 Restore

SCPI Command :

```
DIAGnostic:ACCess:RESTore
```

class RestoreCls

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

set() → None

```
# SCPI: DIAGnostic:ACCess:RESTore
driver.diagnostic.access.restore.set()
```

No command help available

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

```
# SCPI: DIAGnostic:ACCess:RESTore
driver.diagnostic.access.restore.set_with_opc()
```

No command help available

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

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

6.7.1.2 Scenario

SCPI Command :

```
DIAGnostic:ACCess:SCENario
```

class ScenarioCls

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

set(*enabled: List[bool] = None*, *instruments_mask: List[int] = None*, *use_case: List[str] = None*) → None

```
# SCPI: DIAGnostic:ACCess:SCENario
driver.diagnostic.access.scenario.set(enabled = [True, False, True],␣
↪instruments_mask = [1, 2, 3], use_case = ['abc1', 'abc2', 'abc3'])
```

No command help available

param enabled
No help available

param instruments_mask
No help available

param use_case
No help available

6.7.2 BgInfo

SCPI Command :

```
DIAGnostic:BGINfo:CATalog
```

class BgInfoCls

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

get_catalog() → List[str]

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

No command help available

return
boards: No help available

6.7.3 Cmw<CmwVariant>

RepCap Settings

```
# Range: Cmw1 .. Cmw100
rc = driver.diagnostic.cmw.repcap_cmwVariant_get()
driver.diagnostic.cmw.repcap_cmwVariant_set(repcap.CmwVariant.Cmw1)
```

class CmwCls

Cmw commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: CmwVariant, default value after init: CmwVariant.Cmw1

Cloning the Group

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

Subgroups

6.7.3.1 LedTest

class LedTestCls

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

Cloning the Group

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

Subgroups

6.7.3.1.1 Rx

SCPI Command :

```
DIAGnostic:CMW<variant>:LEDTest:RX
```

class RxCls

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

set(*test: bool*, *cmwVariant=CmwVariant.Default*) → None

```
# SCPI: DIAGnostic:CMW<variant>:LEDTest:RX
driver.diagnostic.cmw.ledTest.rx.set(test = False, cmwVariant = repcap.
↪CmwVariant.Default)
```

No command help available

param test
No help available

param cmwVariant
optional repeated capability selector. Default value: Cmw1 (settable in the interface 'Cmw')

6.7.3.1.2 Tx

SCPI Command :

```
DIAGnostic:CMW<variant>:LEDTest:TX
```

class TxCls

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

set(*test: bool*, *cmwVariant=CmwVariant.Default*) → None

```
# SCPI: DIAGnostic:CMW<variant>:LEDTest:TX
driver.diagnostic.cmw.ledTest.tx.set(test = False, cmwVariant = repcap.
↪CmwVariant.Default)
```

No command help available

param test
No help available

param cmwVariant
optional repeated capability selector. Default value: Cmw1 (settable in the interface 'Cmw')

6.7.4 Compass

SCPI Commands :

```
DIAGnostic:COMPass:VERSion
DIAGnostic:COMPass:HEAPcheck
```

class CompassCls

Compass commands group definition. 12 total commands, 3 Subgroups, 2 group commands

get_heap_check() → bool

```
# SCPI: DIAGnostic:COMPass:HEAPcheck
value: bool = driver.diagnostic.compass.get_heap_check()
```

No command help available

return
enable: No help available

get_version() → str

```
# SCPI: DIAGnostic:COMPass:VERSion
value: str = driver.diagnostic.compass.get_version()
```

No command help available

return
version: No help available

set_heap_check(*enable: bool*) → None

```
# SCPI: DIAGnostic:COMPass:HEAPcheck
driver.diagnostic.compass.set_heap_check(enable = False)
```

No command help available

param enable
No help available

Cloning the Group

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

Subgroups

6.7.4.1 Dbase

class DbaseCls

Dbase commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.4.1.1 Rlogging

SCPI Commands :

```
DIAGnostic:COMPass:DBASe:RLOGging:MODE
DIAGnostic:COMPass:DBASe:RLOGging:DEVice
DIAGnostic:COMPass:DBASe:RLOGging:CLEar
```

class RloggingCls

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

clear() → None

```
# SCPI: DIAGnostic:COMPass:DBASe:RLOGging:CLEar
driver.diagnostic.compass.dbase.rlogging.clear()
```

No command help available

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

```
# SCPI: DIAGnostic:COMPass:DBASe:RLOGging:CLEar
driver.diagnostic.compass.dbase.rlogging.clear_with_opc()
```

No command help available

Same as clear, but waits for the operation to complete before continuing further. Use the RsCmwBase.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_device() → DiagLoggingDevice

```
# SCPI: DIAGnostic:COMPass:DBASe:RLOGging:DEVice
value: enums.DiagLoggingDevice = driver.diagnostic.compass.dbase.rlogging.get_
↪device()
```

No command help available

return
arg_0: No help available

get_mode() → DiagLoggigMode

```
# SCPI: DIAGnostic:COMPass:DBASe:RLOGging:MODE
value: enums.DiagLoggigMode = driver.diagnostic.compass.dbase.rlogging.get_
↪mode()
```

No command help available

return
arg_0: No help available

set_device(*arg_0: DiagLoggingDevice*) → None

```
# SCPI: DIAGnostic:COMPass:DBASe:RLOGging:DEVice
driver.diagnostic.compass.dbase.rlogging.set_device(arg_0 = enums.
↪DiagLoggingDevice.ALL)
```

No command help available

param arg_0
No help available

set_mode(*arg_0: DiagLoggigMode*) → None

```
# SCPI: DIAGnostic:COMPass:DBASe:RLOGging:MODE
driver.diagnostic.compass.dbase.rlogging.set_mode(arg_0 = enums.DiagLoggigMode.
↪DETailed)
```

No command help available

param arg_0
No help available

Cloning the Group

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

Subgroups

6.7.4.1.1.1 Protocol

SCPI Command :

```
DIAGnostic:COMPass:DBASe:RLOGging:PROTocol
```

class ProtocolCls

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

get(*file: str*) → str

```
# SCPI: DIAGnostic:COMPass:DBASe:RLOGging:PROTocol
value: str = driver.diagnostic.compass.dbase.rlogging.protocol.get(file = 'abc')
```

No command help available

param file
No help available

return
protocol: No help available

6.7.4.1.2 TaLogging

SCPI Commands :

```
DIAGnostic:COMPass:DBASe:TALogging:CLEar
DIAGnostic:COMPass:DBASe:TALogging:DEVice
```

class TaLoggingCls

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

clear() → None

```
# SCPI: DIAGnostic:COMPass:DBASe:TALogging:CLEar
driver.diagnostic.compass.dbase.taLogging.clear()
```

No command help available

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

```
# SCPI: DIAGnostic:COMPass:DBASe:TALogging:CLEar
driver.diagnostic.compass.dbase.taLogging.clear_with_opc()
```

No command help available

Same as clear, but waits for the operation to complete before continuing further. Use the RsCmwBase.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_device() → DiagLoggingDevice

```
# SCPI: DIAGnostic:COMPass:DBASe:TALogging:DEVice
value: enums.DiagLoggingDevice = driver.diagnostic.compass.dbase.taLogging.get_
↪device()
```

No command help available

return
device: No help available

set_device(*device: DiagLoggingDevice*) → None

```
# SCPI: DIAGnostic:COMPass:DBASe:TALogging:DEVice
driver.diagnostic.compass.dbase.taLogging.set_device(device = enums.
↪DiagLoggingDevice.ALL)
```

No command help available

param device
No help available

Cloning the Group

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

Subgroups

6.7.4.1.2.1 Mode

SCPI Command :

```
DIAGnostic:COMPass:DBASe:TALogging:MODE
```

class ModeCls

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

get(*name: str*) → DiagLoggigMode

```
# SCPI: DIAGnostic:COMPass:DBASe:TALogging:MODE
value: enums.DiagLoggigMode = driver.diagnostic.compass.dbase.taLogging.mode.
↪get(name = 'abc')
```

No command help available

param name
No help available

return
mod: No help available

set(*name: str*, *mod: DiagLoggigMode*) → None

```
# SCPI: DIAGnostic:COMPass:DBASe:TALogging:MODE
driver.diagnostic.compass.dbase.taLogging.mode.set(name = 'abc', mod = enums.
↪DiagLoggigMode.DETailed)
```

No command help available

param name
No help available

param mod
No help available

6.7.4.1.2.2 Protocol

SCPI Command :

```
DIAGnostic:COMPass:DBASe:TALogging:PROTocol
```

class ProtocolCls

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

get(*file: str*) → int

```
# SCPI: DIAGnostic:COMPass:DBASe:TALogging:PROTocol
value: int = driver.diagnostic.compass.dbase.taLogging.protocol.get(file = 'abc
↪')
```

No command help available

param file
No help available

return
result: No help available

6.7.4.2 Debug

SCPI Command :

```
DIAGnostic:COMPass:DEBug:MODE
```

class DebugCls

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

get_mode() → bool

```
# SCPI: DIAGnostic:COMPass:DEBug:MODE
value: bool = driver.diagnostic.compass.debug.get_mode()
```

No command help available

return
debug_mode: No help available

set_mode(*debug_mode: bool*) → None

```
# SCPI: DIAGnostic:COMPass:DEBug:MODE
driver.diagnostic.compass.debug.set_mode(debug_mode = False)
```

No command help available

param debug_mode
No help available

6.7.4.3 Statistics

class StatisticsCls

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

Cloning the Group

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

Subgroups

6.7.4.3.1 Process

SCPI Command :

```
DIAGnostic:COMPass:STATistics:PROCess
```

class ProcessCls

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

get(*token: str*) → str

```
# SCPI: DIAGnostic:COMPass:STATistics:PROCess
value: str = driver.diagnostic.compass.statistics.process.get(token = 'abc')
```

No command help available

param token
No help available

return
statistics: No help available

6.7.5 Eeprom

class EepromCls

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

Cloning the Group

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

Subgroups

6.7.5.1 Data

SCPI Command :

```
DIAGnostic:EEPRom:DATA
```

class DataCls

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

get() → str

```
# SCPI: DIAGnostic:EEPRom:DATA
value: str = driver.diagnostic.eeprom.data.get()
```

No command help available

return
data_folder: No help available

set(*board: str*, *table_id: int*, *board_instance: int = None*) → None

```
# SCPI: DIAGnostic:EEPRom:DATA
driver.diagnostic.eeprom.data.set(board = 'abc', table_id = 1, board_instance =␣
↪1)
```

No command help available

param board
No help available

param table_id
No help available

param board_instance
No help available

6.7.5.2 Header

SCPI Command :

```
DIAGnostic:EEPRom:HEADer
```

class HeaderCls

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

get() → str

```
# SCPI: DIAGnostic:EEPRom:HEADer
value: str = driver.diagnostic.eeprom.header.get()
```

No command help available

return
header: No help available

set(*board: str*, *board_instance: int = None*) → None

```
# SCPI: DIAGnostic:EEPRom:HEADer
driver.diagnostic.eeprom.header.set(board = 'abc', board_instance = 1)
```

No command help available

param board
No help available

param board_instance
No help available

6.7.6 Error

class ErrorCls

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

Cloning the Group

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

Subgroups

6.7.6.1 Queue

SCPI Commands :

```
DIAGnostic:ERRor:QUEue:SIZE
DIAGnostic:ERRor:QUEue:LENGth
```

class QueueCls

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

get_length() → int

```
# SCPI: DIAGnostic:ERRor:QUEue:LENGth
value: int = driver.diagnostic.error.queue.get_length()
```

No command help available

return
text_length: No help available

get_size() → int

```
# SCPI: DIAGnostic:ERRor:QUEue:SIZE
value: int = driver.diagnostic.error.queue.get_size()
```

No command help available

return
size: No help available

set_length(*text_length: int*) → None

```
# SCPI: DIAGnostic:ERRor:QUEue:LENGth
driver.diagnostic.error.queue.set_length(text_length = 1)
```

No command help available

param text_length
No help available

set_size(*size: int*) → None

```
# SCPI: DIAGnostic:ERRor:QUEue:SIZE
driver.diagnostic.error.queue.set_size(size = 1)
```

No command help available

param size
No help available

Cloning the Group

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

Subgroups

6.7.6.1.1 Push

SCPI Command :

```
DIAGnostic:ERRor:QUEue:PUSH
```

class PushCls

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

set(*code: int*, *text: str*, *guid: str*, *info: str*) → None

```
# SCPI: DIAGnostic:ERRor:QUEue:PUSH
driver.diagnostic.error.queue.push.set(code = 1, text = 'abc', guid = 'abc',␣
↪info = 'abc')
```

No command help available

param code
No help available

param text
No help available

param guid
No help available

param info
No help available

6.7.7 FootPrint

class FootPrintCls

FootPrint commands group definition. 8 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.7.1 Element

SCPI Command :

```
DIAGnostic:FOOTprint:ELEMent:IDS
```

class ElementCls

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

get_ids() → List[int]

```
# SCPI: DIAGnostic:FOOTprint:ELEMent:IDS
value: List[int] = driver.diagnostic.footPrint.element.get_ids()
```

No command help available

return
element_ids: No help available

Cloning the Group

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

Subgroups

6.7.7.1.1 Connection

class ConnectionCls

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

Cloning the Group

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

Subgroups

6.7.7.1.1.1 Target

class TargetCls

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

Cloning the Group

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

Subgroups

6.7.7.1.1.2 Ids

SCPI Command :

```
DIAGnostic:FOOTprint:ELEMent:CONNection:TARGet:IDS
```

class IdsCls

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

get(*ielement_id: float*) → str

```
# SCPI: DIAGnostic:FOOTprint:ELEMent:CONNection:TARGet:IDS
value: str = driver.diagnostic.footPrint.element.connection.target.ids.
↪get(ielement_id = 1.0)
```

No command help available

param ielement_id
No help available

return
starget_ids: No help available

6.7.7.1.2 Data

SCPI Command :

```
DIAGnostic:FOOTprint:ELEMent:DATA
```

class DataCls

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

class GetStruct

Response structure. Fields:

- Stype_Name: str: No parameter help available
- Spart_Name: str: No parameter help available
- Efunctionality: int: No parameter help available

get(*ielement_id: float*) → GetStruct

```
# SCPI: DIAGnostic:FOOTprint:ELEMent:DATA
value: GetStruct = driver.diagnostic.footPrint.element.data.get(ielement_id = 1.
↪0)
```

No command help available

param ielement_id
No help available

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

6.7.7.1.3 Properties

SCPI Command :

```
DIAGnostic:FOOTprint:ELEMent:PROPerties
```

class PropertiesCls

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

get(*ielement_id: float*, *bnormalized: bool*) → str

```
# SCPI: DIAGnostic:FOOTprint:ELEMent:PROPerties
value: str = driver.diagnostic.footPrint.element.properties.get(ielement_id = 1.
↪0, bnormalized = False)
```

No command help available

param ielement_id
No help available

param bnormalized
No help available

return
sproperties: No help available

6.7.7.1.4 References

SCPI Command :

```
DIAGnostic:FOOTprint:ELEMent:REFerences
```

class ReferencesCls

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

get(*ielement_id: float*, *sreference_type: str*) → List[int]

```
# SCPI: DIAGnostic:FOOTprint:ELEMent:REFerences
value: List[int] = driver.diagnostic.footPrint.element.references.get(ielement_
↪id = 1.0, sreference_type = 'abc')
```

No command help available

param ielement_id
No help available

param sreference_type
No help available

return
ielement_ids: No help available

6.7.7.2 Li

class LiCls

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

Cloning the Group

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

Subgroups

6.7.7.2.1 Usecases

SCPI Command :

```
DIAGnostic:FOOTprint:LI:USECases
```

class UsecasesCls

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

get(*ili_id: int*) → str

```
# SCPI: DIAGnostic:FOOTprint:LI:USECases
value: str = driver.diagnostic.footPrint.li.usecases.get(ili_id = 1)
```

No command help available

param ili_id
No help available

return
the_result: No help available

6.7.7.3 UseCase

SCPI Command :

```
DIAGnostic:FOOTprint:USECase:IDS
```

class UseCaseCls

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

get_ids() → List[int]

```
# SCPI: DIAGnostic:FOOTprint:USECase:IDS
value: List[int] = driver.diagnostic.footPrint.useCase.get_ids()
```

No command help available

return
ids: No help available

Cloning the Group

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

Subgroups

6.7.7.3.1 Data

SCPI Command :

```
DIAGnostic:FOOTprint:USECase:DATA
```

class DataCls

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

class GetStruct

Response structure. Fields:

- Sname: str: No parameter help available
- Efunctionality: int: No parameter help available

get(*iuse_case_id: int*) → GetStruct

```
# SCPI: DIAGnostic:FOOTprint:USECase:DATA
value: GetStruct = driver.diagnostic.footPrint.useCase.data.get(iuse_case_id =␣
↪1)
```

No command help available

param iuse_case_id
No help available

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

6.7.8 Help

class HelpCls

Help commands group definition. 5 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.8.1 Headers

SCPI Command :

```
DIAGnostic:HELP:HEADers
```

class HeadersCls

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

get_value() → bytes

```
# SCPI: DIAGnostic:HELP:HEADers
value: bytes = driver.diagnostic.help.headers.get_value()
```

No command help available

return
header: No help available

Cloning the Group

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

Subgroups

6.7.8.1.1 Access

SCPI Commands :

```
DIAGnostic:HELP:HEADers:ACCess:ENABled
DIAGnostic:HELP:HEADers:ACCess:DENied
```

class AccessCls

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

class DeniedStruct

Structure for reading output parameters. Fields:

- Count: int: No parameter help available
- Header: List[str]: No parameter help available

class EnabledStruct

Structure for reading output parameters. Fields:

- Count: int: No parameter help available
- Header: List[str]: No parameter help available

get_denied() → DeniedStruct

```
# SCPI: DIAGnostic:HELP:HEADers:ACCess:DENied
value: DeniedStruct = driver.diagnostic.help.headers.access.get_denied()
```

No command help available

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

get_enabled() → EnabledStruct

```
# SCPI: DIAGnostic:HELP:HEADers:ACCess:ENABled
value: EnabledStruct = driver.diagnostic.help.headers.access.get_enabled()
```

No command help available

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

6.7.8.2 Syntax

SCPI Commands :

```
DIAGnostic:HELP:SYNTax
DIAGnostic:HELP:SYNTax:ALL
```

class SyntaxCls

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

get(*header: str*) → str

```
# SCPI: DIAGnostic:HELP:SYNTax
value: str = driver.diagnostic.help.syntax.get(header = 'abc')
```

No command help available

param header
No help available

return
syntax: No help available

get_all() → bytes

```
# SCPI: DIAGnostic:HELP:SYNTax:ALL
value: bytes = driver.diagnostic.help.syntax.get_all()
```

No command help available

return
commands: No help available

6.7.9 Instrument

SCPI Commands :

```
DIAGnostic:INSTrument:LOAD
DIAGnostic:INSTrument:UNLoad
```

class InstrumentCls

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

load(*appl_name_and_li_number: str*) → None

```
# SCPI: DIAGnostic:INSTrument:LOAD
driver.diagnostic.instrument.load(appl_name_and_li_number = 'abc')
```

No command help available

param appl_name_and_li_number
No help available

set_unload(*appl_name_and_li_number: str*) → None

```
# SCPI: DIAGnostic:INSTrument:UNLoad
driver.diagnostic.instrument.set_unload(appl_name_and_li_number = 'abc')
```

No command help available

param appl_name_and_li_number
No help available

6.7.10 Kremote

class KremoteCls

Kremote commands group definition. 9 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.10.1 Tmonitor

SCPI Command :

```
DIAGnostic:KREMote:TMONitor:RESet
```

class TmonitorCls

Tmonitor commands group definition. 9 total commands, 4 Subgroups, 1 group commands

reset() → None

```
# SCPI: DIAGnostic:KREMote:TMONitor:RESet
driver.diagnostic.kremote.tmonitor.reset()
```

No command help available

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

```
# SCPI: DIAGnostic:KREMote:TMONitor:RESet
driver.diagnostic.kremote.tmonitor.reset_with_opc()
```

No command help available

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

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

Cloning the Group

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

Subgroups

6.7.10.1.1 Dump

SCPI Command :

```
DIAGnostic:KREMote:TMONitor:DUMP
```

class DumpCls

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

get(*formatting: TextFormatting = None*) → bytes

```
# SCPI: DIAGnostic:KREMote:TMONitor:DUMP
value: bytes = driver.diagnostic.kremote.tmonitor.dump.get(formatting = enums.
↪TextFormatting.TXT)
```

No command help available

param formatting
No help available

return
trace_report: No help available

6.7.10.1.2 Enable

SCPI Commands :

```
DIAGnostic:KREMote:TMONitor:ENABle:STATistic
DIAGnostic:KREMote:TMONitor:ENABle:TIMing
DIAGnostic:KREMote:TMONitor:ENABle:TRACe
DIAGnostic:KREMote:TMONitor:ENABle:RPC
DIAGnostic:KREMote:TMONitor:ENABle
```

class EnableCls

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

get_rpc() → bool

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:RPC
value: bool = driver.diagnostic.kremote.tmonitor.enable.get_rpc()
```

No command help available

return
rpc_enable: No help available

get_statistic() → bool

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:STATistic
value: bool = driver.diagnostic.kremote.tmonitor.enable.get_statistic()
```

No command help available

return
performance_mode: No help available

get_timing() → bool

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:TIMing
value: bool = driver.diagnostic.kremote.tmonitor.enable.get_timing()
```

No command help available

return
timing_enable: No help available

get_trace() → bool

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:TRACe
value: bool = driver.diagnostic.kremote.tmonitor.enable.get_trace()
```

No command help available

return
trace_mode: No help available

get_value() → bool

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle
value: bool = driver.diagnostic.kremote.tmonitor.enable.get_value()
```

No command help available

return
remote_diagnose: No help available

set_rpc(*rpc_enable: bool*) → None

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:RPC
driver.diagnostic.kremote.tmonitor.enable.set_rpc(rpc_enable = False)
```

No command help available

param rpc_enable
No help available

set_statistic(*performance_mode: bool*) → None

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:STATistic
driver.diagnostic.kremote.tmonitor.enable.set_statistic(performance_mode =␣
↪False)
```

No command help available

param performance_mode
No help available

set_timing(*timing_enable: bool*) → None

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:TIMing
driver.diagnostic.kremote.tmonitor.enable.set_timing(timing_enable = False)
```

No command help available

param timing_enable
No help available

set_trace(*trace_mode: bool*) → None

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle:TRACe
driver.diagnostic.kremote.tmonitor.enable.set_trace(trace_mode = False)
```

No command help available

param trace_mode
No help available

set_value(*remote_diagnose: bool*) → None

```
# SCPI: DIAGnostic:KREMote:TMONitor:ENABle
driver.diagnostic.kremote.tmonitor.enable.set_value(remote_diagnose = False)
```

No command help available

param remote_diagnose
No help available

6.7.10.1.3 Statistic

SCPI Command :

```
DIAGnostic:KREMote:TMONitor:STATistic
```

class StatisticCls

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

get(*formatting: TextFormatting = None*) → bytes

```
# SCPI: DIAGnostic:KREMote:TMONitor:STATistic
value: bytes = driver.diagnostic.kremote.tmonitor.statistic.get(formatting =␣
↪enums.TextFormatting.TXT)
```

No command help available

param formatting
No help available

return
coverage_report: No help available

6.7.10.1.4 Trace

SCPI Command :

```
DIAGnostic:KREMote:TMONitor:TRACe
```

class TraceCls

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

get(*formatting: TextFormatting = None*) → bytes

```
# SCPI: DIAGnostic:KREMote:TMONitor:TRACe
value: bytes = driver.diagnostic.kremote.tmonitor.trace.get(formatting = enums.
↪TextFormatting.TXT)
```

No command help available

param formatting
No help available

return
trace_report: No help available

6.7.11 Log

class LogCls

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

Cloning the Group

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

Subgroups

6.7.11.1 Dump

SCPI Command :

```
DIAGnostic:LOG:DUMP
```

class DumpCls

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

set() → None

```
# SCPI: DIAGnostic:LOG:DUMP
driver.diagnostic.log.dump.set()
```

No command help available

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

```
# SCPI: DIAGnostic:LOG:DUMP
driver.diagnostic.log.dump.set_with_opc()
```

No command help available

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

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

6.7.12 Pias

SCPI Commands :

```
DIAGnostic:PIAS:HOST
DIAGnostic:PIAS:ID
```

class PiasCls

Pias commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_host() → str

```
# SCPI: DIAGnostic:PIAS:HOST
value: str = driver.diagnostic.pias.get_host()
```

No command help available

return
hostname: No help available

get_id() → str

```
# SCPI: DIAGnostic:PIAS:ID
value: str = driver.diagnostic.pias.get_id()
```

No command help available

return
pias_id: No help available

Cloning the Group

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

Subgroups

6.7.12.1 Connect

SCPI Command :

```
DIAGnostic:PIAS:CONNect
```

class ConnectCls

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

get(*handle: str*, *pias_id: str*) → int

```
# SCPI: DIAGnostic:PIAS:CONNect
value: int = driver.diagnostic.pias.connect.get(handle = 'abc', pias_id = 'abc')
```

No command help available

param handle
No help available

param pias_id
No help available

return
result: No help available

Cloning the Group

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

Subgroups

6.7.12.1.1 Multiple

SCPI Command :

```
DIAGnostic:PIAS:CONNect:MULTiple
```

class MultipleCls

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

get(*handle: List[str]*) → int

```
# SCPI: DIAGnostic:PIAS:CONNect:MULTiple
value: int = driver.diagnostic.pias.connect.multiple.get(handle = ['abc1', 'abc2
↪', 'abc3'])
```

No command help available

param handle
No help available

return
result: No help available

6.7.12.2 Scan

SCPI Command :

```
DIAGnostic:PIAS:SCAN
```

class ScanCls

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

get(*timeout: int*, *subnet: SubnetScope*, *device_group_count: int*, *ids: List[str]*) → str

```
# SCPI: DIAGnostic:PIAS:SCAN
value: str = driver.diagnostic.pias.scan.get(timeout = 1, subnet = enums.
↪SubnetScope.ALL, device_group_count = 1, ids = ['abc1', 'abc2', 'abc3'])
```

No command help available

param timeout
No help available

param subnet
No help available

param device_group_count
No help available

param ids
No help available

return
result: No help available

6.7.13 Product

SCPI Commands :

```
DIAGnostic:PRODuct:ID
DIAGnostic:PRODuct:DESCription
DIAGnostic:PRODuct:CATalog
DIAGnostic:PRODuct:SELect
DIAGnostic:PRODuct:GROup
```

class ProductCls

Product commands group definition. 9 total commands, 2 Subgroups, 5 group commands

class IdStruct

Structure for reading output parameters. Fields:

- Manufacturer: str: No parameter help available
- Device_Id: str: No parameter help available
- Version: str: No parameter help available

get_catalog() → str

```
# SCPI: DIAGnostic:PRODuct:CATalog
value: str = driver.diagnostic.product.get_catalog()
```

No command help available

return
material_number: No help available

get_description() → bytes

```
# SCPI: DIAGnostic:PRODuct:DESCription
value: bytes = driver.diagnostic.product.get_description()
```

No command help available

return
product_description: No help available

get_group() → str

```
# SCPI: DIAGnostic:PRODuct:GROup
value: str = driver.diagnostic.product.get_group()
```

No command help available

return
group: No help available

get_id() → IdStruct

```
# SCPI: DIAGnostic:PRODuct:ID
value: IdStruct = driver.diagnostic.product.get_id()
```

No command help available

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

get_select() → str

```
# SCPI: DIAGnostic:PRODuct:SELect
value: str = driver.diagnostic.product.get_select()
```

No command help available

return
material_number: No help available

set_group(*group: str*) → None

```
# SCPI: DIAGnostic:PRODuct:GROup
driver.diagnostic.product.set_group(group = 'abc')
```

No command help available

param group
No help available

set_select(*material_number: str*) → None

```
# SCPI: DIAGnostic:PRODuct:SELect
driver.diagnostic.product.set_select(material_number = 'abc')
```

No command help available

param material_number
No help available

Cloning the Group

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

Subgroups

6.7.13.1 MacAddress

SCPI Commands :

```
DIAGnostic:PRODuct:MACaddress:STORe
DIAGnostic:PRODuct:MACaddress:RESTore
DIAGnostic:PRODuct:MACaddress
```

class MacAddressCls

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

get_value() → str

```
# SCPI: DIAGnostic:PRODuct:MACaddress
value: str = driver.diagnostic.product.macAddress.get_value()
```

No command help available

return
mac_addr: No help available

set_restore(*source: StoragePlace*) → None

```
# SCPI: DIAGnostic:PRODuct:MACaddress:RESTore
driver.diagnostic.product.macAddress.set_restore(source = enums.StoragePlace.
↪EEPRom)
```

No command help available

param source
No help available

set_store(*target: StoragePlace*) → None

```
# SCPI: DIAGnostic:PRODuct:MACaddress:STORe
driver.diagnostic.product.macAddress.set_store(target = enums.StoragePlace.
↪EEPRom)
```

No command help available

param target
No help available

6.7.13.2 Time

SCPI Command :

```
DIAGnostic:PRODuct:TIME:OPERating
```

class TimeCls

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

get_operating() → str

```
# SCPI: DIAGnostic:PRODuct:TIME:OPERating
value: str = driver.diagnostic.product.time.get_operating()
```

No command help available

return
operating_time: No help available

6.7.14 Record

class RecordCls

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

Cloning the Group

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

Subgroups

6.7.14.1 Macro

class MacroCls

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

Cloning the Group

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

Subgroups

6.7.14.1.1 File

SCPI Commands :

```
DIAGnostic:RECord:MACRo:FILE:SIZE
DIAGnostic:RECord:MACRo:FILE:FILTer
```

class FileCls

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

class FilterPyStruct

Structure for setting input parameters. Fields:

- Binput: bool: No parameter help available

- Boutput: bool: No parameter help available
- Berror: bool: No parameter help available
- Btrigger: bool: No parameter help available
- Bdevice_Clear: bool: No parameter help available
- Bstatus_Register: bool: No parameter help available
- Bconnection: bool: No parameter help available
- Bremote_Local_Events: bool: No parameter help available

get_filter_py() → FilterPyStruct

```
# SCPI: DIAGnostic:RECord:MACRo:FILE:FILTer
value: FilterPyStruct = driver.diagnostic.record.macro.file.get_filter_py()
```

No command help available

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

get_size() → int

```
# SCPI: DIAGnostic:RECord:MACRo:FILE:SIZE
value: int = driver.diagnostic.record.macro.file.get_size()
```

No command help available

return
ifile_size: No help available

set_filter_py(*value: FilterPyStruct*) → None

```
# SCPI: DIAGnostic:RECord:MACRo:FILE:FILTer
structure = driver.diagnostic.record.macro.file.FilterPyStruct()
structure.Binput: bool = False
structure.Boutput: bool = False
structure.Berror: bool = False
structure.Btrigger: bool = False
structure.Bdevice_Clear: bool = False
structure.Bstatus_Register: bool = False
structure.Bconnection: bool = False
structure.Bremote_Local_Events: bool = False
driver.diagnostic.record.macro.file.set_filter_py(value = structure)
```

No command help available

param value
see the help for FilterPyStruct structure arguments.

set_size(*ifile_size: int*) → None

```
# SCPI: DIAGnostic:RECord:MACRo:FILE:SIZE
driver.diagnostic.record.macro.file.set_size(ifile_size = 1)
```

No command help available

param ifile_size
No help available

6.7.15 Routing

SCPI Command :

```
DIAGnostic:ROUTing:CATalog
```

class RoutingCls

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

get_catalog() → List[str]

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

No command help available

return
routing_name: No help available

Cloning the Group

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

Subgroups

6.7.15.1 Expert

class ExpertCls

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

Cloning the Group

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

Subgroups

6.7.15.1.1 Setup

SCPI Command :

```
DIAGnostic:ROUTing:EXPert:SETup
```

class SetupCls

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

get(*routing_name: str*) → List[ExpertSetup]

```
# SCPI: DIAGnostic:ROUTing:EXPert:SETup
value: List[enums.ExpertSetup] = driver.diagnostic.routing.expert.setup.
↪get(routing_name = 'abc')
```

No command help available

param routing_name
No help available

return
data: No help available

set(*routing_name: str*, *data: List[ExpertSetup]*) → None

```
# SCPI: DIAGnostic:ROUTing:EXPert:SETup
driver.diagnostic.routing.expert.setup.set(routing_name = 'abc', data =␣
↪[ExpertSetup.BBG1, ExpertSetup.SUW7])
```

No command help available

param routing_name
No help available

param data
No help available

6.7.16 SingleCmw

SCPI Command :

```
DIAGnostic:CMWS:LEDTest
```

class SingleCmwCls

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

set_led_test(*test: bool*) → None

```
# SCPI: DIAGnostic:CMWS:LEDTest
driver.diagnostic.singleCmw.set_led_test(test = False)
```

No command help available

param test
No help available

6.7.17 Status

SCPI Command :

```
DIAGnostic:STATus:OPC
```

class StatusCls

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

class OpcStruct

Structure for reading output parameters. Fields:

- Opc_Counter: int: No parameter help available
- Opc_Active_Counter: int: No parameter help available
- Opc_State: bool: No parameter help available
- Opc_Command_State: bool: No parameter help available
- Opc_Query_State: bool: No parameter help available

get_opc() → OpcStruct

```
# SCPI: DIAGnostic:STATus:OPC
value: OpcStruct = driver.diagnostic.status.get_opc()
```

No command help available

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

6.8 Display

SCPI Command :

```
DISPlay:FORMat
```

class DisplayCls

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

get_format_py() → str

```
# SCPI: DISPlay:FORMat
value: str = driver.display.get_format_py()
```

No command help available

return
arg_0: No help available

set_format_py(*arg_0: str*) → None

```
# SCPI: DISPlay:FORMat
driver.display.set_format_py(arg_0 = rawAbc)
```

No command help available

param arg_0
No help available

Cloning the Group

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

Subgroups

6.8.1 Window<Window>

RepCap Settings

```
# Range: Win1 .. Win32
rc = driver.display.window.repcap_window_get()
driver.display.window.repcap_window_set(repcap.Window.Win1)
```

class WindowCls

Window commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Window, default value after init: Window.Win1

Cloning the Group

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

Subgroups

6.8.1.1 Select

SCPI Command :

```
DISPlay[:WINDow<1-n>]:SELect
```

class SelectCls

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

set(*window=Window.Default*) → None

```
# SCPI: DISPlay[:WINDow<1-n>]:SELect
driver.display.window.select.set(window = repcap.Window.Default)
```

No command help available

param window
optional repeated capability selector. Default value: Win1 (settable in the interface 'Window')

set_with_opc(*window=Window.Default*, *opc_timeout_ms: int = -1*) → None

6.9 FirmwareUpdate

SCPI Command :

```
FETCh:FWUPdate:VERSions
```

class FirmwareUpdateCls

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

get_versions() → List[str]

```
# SCPI: FETCh:FWUPdate:VERSions
value: List[str] = driver.firmwareUpdate.get_versions()
```

No command help available

return
versions: No help available

6.10 FormatPy

SCPI Commands :

```
FORMat:BASE:BORDer
FORMat:BASE:DINTerchange
FORMat:BASE:SREGister
```

class FormatPyCls

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

get_border() → ByteOrder

```
# SCPI: FORMat:BASE:BORDer
value: enums.ByteOrder = driver.formatPy.get_border()
```

No command help available

return
byte_order: No help available

get_dinterchange() → bool

```
# SCPI: FORMat:BASE:DINTerchange
value: bool = driver.formatPy.get_dinterchange()
```

No command help available

return
dif_format: No help available

get_sregister() → StatRegFormat

```
# SCPI: FORMat:BASE:SREGister
value: enums.StatRegFormat = driver.formatPy.get_sregister()
```

No command help available

return
status_register_format: No help available

set_border(*byte_order: ByteOrder*) → None

```
# SCPI: FORMat:BASE:BORDer
driver.formatPy.set_border(byte_order = enums.ByteOrder.NORMal)
```

No command help available

param byte_order
No help available

set_dinterchange(*dif_format: bool*) → None

```
# SCPI: FORMat:BASE:DINTerchange
driver.formatPy.set_dinterchange(dif_format = False)
```

No command help available

param dif_format
No help available

set_sregister(*status_register_format: StatRegFormat*) → None

```
# SCPI: FORMat:BASE:SREGister
driver.formatPy.set_sregister(status_register_format = enums.StatRegFormat.
↪ASCii)
```

No command help available

param status_register_format
No help available

Cloning the Group

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

Subgroups

6.10.1 Data

SCPI Command :

```
FORMat:BASE[:DATA]
```

class DataCls

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

class DataStruct

Response structure. Fields:

- Data_Type: enums.DataFormat: ASCii | REAL | BINary | HEXadecimal | OCTal ASCii Numeric data is transferred as ASCII bytes. Floating point numbers are transferred in scientific E notation. REAL Numeric data is transferred in a definite length block as IEEE floating point numbers (block data) . BINary | HEXadecimal | OCTal Numeric data is transferred in binary, hexadecimal or octal format.
- Data_Length: int: The meaning depends on the DataType as listed below. A zero returned by a query means that the default value is used. For ASCii Decimal places of floating point numbers. That means, number of 'b' digits in the scientific notation a.bbbbbbE+ccc. Default: six decimal places For REAL Length of floating point numbers in bits: 32 bits = 4 bytes, format #14… 64 bits = 8 bytes, format #18… Default: 64 bits For BINary, HEXadecimal, OCTal Minimum number of digits. If the number is longer, more digits are used. If it is shorter, leading zeros are added. Default: 0, no leading zeros

get() → DataStruct

```
# SCPI: FORMat:BASE[:DATA]
value: DataStruct = driver.formatPy.data.get()
```

Selects the format for numeric data transferred to and from the R&S CMW, for example query results.

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

set(*data_type: DataFormat*, *data_length: int = None*) → None

```
# SCPI: FORMat:BASE[:DATA]
driver.formatPy.data.set(data_type = enums.DataFormat.ASCii, data_length = 1)
```

Selects the format for numeric data transferred to and from the R&S CMW, for example query results.

param data_type
ASCii | REAL | BINary | HEXadecimal | OCTal ASCii Numeric data is transferred as ASCII bytes. Floating point numbers are transferred in scientific E notation. REAL Numeric data is transferred in a definite length block as IEEE floating point numbers (block data) . BINary | HEXadecimal | OCTal Numeric data is transferred in binary, hexadecimal or octal format.

param data_length
The meaning depends on the DataType as listed below. A zero returned by a query means that the default value is used. For ASCii Decimal places of floating point numbers. That means, number of 'b' digits in the scientific notation a.bbbbbbE+ccc. Default: six decimal places For REAL Length of floating point numbers in bits: 32 bits = 4 bytes, format #14… 64 bits = 8 bytes, format #18… Default: 64 bits For BINary,

HEXadecimal, OCTal Minimum number of digits. If the number is longer, more digits are used. If it is shorter, leading zeros are added. Default: 0, no leading zeros

6.11 Get

SCPI Command :

```
GET:XVALues
```

class GetCls

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

get_xvalues() → List[float]

```
# SCPI: GET:XVALues
value: List[float] = driver.get.get_xvalues()
```

No command help available

return
value: No help available

6.12 GlobalClearStatus

SCPI Command :

```
*GCLS
```

class GlobalClearStatusCls

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

set() → None

```
# SCPI: *GCLS
driver.globalClearStatus.set()
```

No command help available

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

```
# SCPI: *GCLS
driver.globalClearStatus.set_with_opc()
```

No command help available

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

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

6.13 GlobalWait

SCPI Command :

```
*GWAI
```

class GlobalWaitCls

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

set() → None

```
# SCPI: *GWAI
driver.globalWait.set()
```

No command help available

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

```
# SCPI: *GWAI
driver.globalWait.set_with_opc()
```

No command help available

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

SCPI Command :

```
*GTL
```

class GotoLocalCls

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

set() → None

```
# SCPI: *GTL
driver.gotoLocal.set()
```

No command help available

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

```
# SCPI: *GTL
driver.gotoLocal.set_with_opc()
```

No command help available

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

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

6.15 HardCopy

SCPI Commands :

```
HCOPy:DATA
HCOPy:FILE
```

class HardCopyCls

HardCopy commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_data() → bytes

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

Captures a screenshot and returns the result in block data format. method RsCmwBase.HardCopy.data captures the entire window, method RsCmwBase.HardCopy.Interior.data only the interior of the window. It is recommended to 'switch on' the display before sending this command, see method RsCmwBase.System.Display.update.

return
data: block Screenshot in block data format.

set_file(*filename: str*) → None

```
# SCPI: HCOPy:FILE
driver.hardCopy.set_file(filename = 'abc')
```

Captures a screenshot and stores it to the specified file. method RsCmwBase.HardCopy.file captures the entire window, method RsCmwBase.HardCopy.Interior.file only the interior of the window. If a 'Remote' dialog is displayed instead of the normal display contents, this command switches on the display before taking a screenshot, and afterwards off again.

param filename
string Absolute path and name of the file. The file name extension is added automatically according to the configured format (see method RsCmwBase.HardCopy.Device.formatPy) . Aliases are allowed (see method RsCmwBase.MassMemory.aliases) . Wildcards are not allowed.

Cloning the Group

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

Subgroups

6.15.1 Device

SCPI Command :

```
HCOPy:DEVice:FORMat
```

class DeviceCls

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

get_format_py() → ScreenshotFormat

```
# SCPI: HCOPy:DEVice:FORMat
value: enums.ScreenshotFormat = driver.hardCopy.device.get_format_py()
```

Specifies the format of screenshots created via the commands method RsCmwBase.HardCopy.file, method RsCmwBase.HardCopy. data, method RsCmwBase.HardCopy.Interior.file or method RsCmwBase.HardCopy.Interior.data.

return
file_formats: No help available

set_format_py(*file_formats: ScreenshotFormat*) → None

```
# SCPI: HCOPy:DEVice:FORMat
driver.hardCopy.device.set_format_py(file_formats = enums.ScreenshotFormat.BMP)
```

Specifies the format of screenshots created via the commands method RsCmwBase.HardCopy.file, method RsCmwBase.HardCopy. data, method RsCmwBase.HardCopy.Interior.file or method RsCmwBase.HardCopy.Interior.data.

param file_formats
BMP | JPG | PNG BMP: Windows bitmap format JPG: JPEG format PNG: PNG format

6.15.2 Interior

SCPI Commands :

```
HCOPy:INTerior:DATA
HCOPy:INTerior:FILE
```

class InteriorCls

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

get_data() → bytes

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

Captures a screenshot and returns the result in block data format. method RsCmwBase.HardCopy.data captures the entire window, method RsCmwBase.HardCopy.Interior.data only the interior of the window. It is recommended to 'switch on' the display before sending this command, see method RsCmwBase.System.Display.update.

return
data: block Screenshot in block data format.

set_file(*filename: str*) → None

```
# SCPI: HCOPy:INTerior:FILE
driver.hardCopy.interior.set_file(filename = 'abc')
```

Captures a screenshot and stores it to the specified file. method RsCmwBase.HardCopy.file captures the entire window, method RsCmwBase.HardCopy.Interior.file only the interior of the window. If a 'Remote' dialog is displayed instead of the normal display contents, this command switches on the display before taking a screenshot, and afterwards off again.

param filename
string Absolute path and name of the file. The file name extension is added automatically according to the configured format (see method RsCmwBase.HardCopy.Device.formatPy) . Aliases are allowed (see method RsCmwBase.MassMemory.aliases) . Wildcards are not allowed.

6.16 Instrument

SCPI Command :

```
INSTrument:NSELect
```

class InstrumentCls

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

get_nselect() → int

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

No command help available

return
arg_0: No help available

set_nselect(*arg_0: int*) → None

```
# SCPI: INSTrument:NSELect
driver.instrument.set_nselect(arg_0 = 1)
```

No command help available

param arg_0
No help available

Cloning the Group

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

Subgroups

6.16.1 Display

SCPI Commands :

```
INSTrument:DISPlay:CAT
INSTrument:DISPlay:MODE
INSTrument:DISPlay:OPEN
INSTrument:DISPlay:CLOSe
INSTrument:DISPlay
```

class DisplayCls

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

close(*item: str*) → None

```
# SCPI: INSTrument:DISPlay:CLOSe
driver.instrument.display.close(item = 'abc')
```

No command help available

param item
No help available

get_cat() → str

```
# SCPI: INSTrument:DISPlay:CAT
value: str = driver.instrument.display.get_cat()
```

No command help available

return
list_py: No help available

get_mode() → DisplayMode

```
# SCPI: INSTrument:DISPlay:MODE
value: enums.DisplayMode = driver.instrument.display.get_mode()
```

No command help available

return
mode: No help available

get_value() → int

```
# SCPI: INSTrument:DISPlay
value: int = driver.instrument.display.get_value()
```

No command help available

return
instr: No help available

open(*item: str*) → None

```
# SCPI: INSTrument:DISPlay:OPEN
driver.instrument.display.open(item = 'abc')
```

No command help available

param item
No help available

set_mode(*mode: DisplayMode*) → None

```
# SCPI: INSTrument:DISPlay:MODE
driver.instrument.display.set_mode(mode = enums.DisplayMode.AUTomatic)
```

No command help available

param mode
No help available

set_value(*instr: int*) → None

```
# SCPI: INSTrument:DISPlay
driver.instrument.display.set_value(instr = 1)
```

No command help available

param instr
No help available

6.16.2 Select

SCPI Command :

```
INSTrument[:SELect]
```

class SelectCls

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

get_value() → str

```
# SCPI: INSTrument[:SELect]
value: str = driver.instrument.select.get_value()
```

No command help available

return
instrument: No help available

set_value(*instrument: str*) → None

```
# SCPI: INSTrument[:SELect]
driver.instrument.select.set_value(instrument = rawAbc)
```

No command help available

param instrument
No help available

Cloning the Group

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

Subgroups

6.16.2.1 Dstrategy

SCPI Command :

```
INSTrument[:SELect]:DSTRategy
```

class DstrategyCls

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

class DstrategyStruct

Response structure. Fields:

- Arg_0: enums.OperationMode: No parameter help available
- Arg_1: enums.DisplayStrategy: No parameter help available

get() → DstrategyStruct

```
# SCPI: INSTrument[:SELect]:DSTRategy
value: DstrategyStruct = driver.instrument.select.dstrategy.get()
```

No command help available

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

set(*arg_0: OperationMode*, *arg_1: DisplayStrategy = None*) → None

```
# SCPI: INSTrument[:SELect]:DSTRategy
driver.instrument.select.dstrategy.set(arg_0 = enums.OperationMode.LOCal, arg_1␣
↪= enums.DisplayStrategy.BYLayout)
```

No command help available

param arg_0
No help available

param arg_1
No help available

6.17 Ipc

SCPI Commands :

```
INITiate:BASE:IPC
ABORt:BASE:IPC
FETCh:BASE:IPC
```

class IpcCls

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

abort() → None

```
# SCPI: ABORt:BASE:IPC
driver.ipc.abort()
```

No command help available

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

```
# SCPI: ABORt:BASE:IPC
driver.ipc.abort_with_opc()
```

No command help available

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

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

fetch() → ResourceState

```
# SCPI: FETCh:BASE:IPC
value: enums.ResourceState = driver.ipc.fetch()
```

No command help available

return
meas_status: No help available

initiate() → None

```
# SCPI: INITiate:BASE:IPC
driver.ipc.initiate()
```

No command help available

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

```
# SCPI: INITiate:BASE:IPC
driver.ipc.initiate_with_opc()
```

No command help available

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

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

Cloning the Group

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

Subgroups

6.17.1 Result

SCPI Command :

```
FETCh:BASE:IPC:RESult
```

class ResultCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Result_Number: int: No parameter help available
- Date: str: No parameter help available
- Result_Text: str: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:BASE:IPC:RESult
value: FetchStruct = driver.ipc.result.fetch()
```

No command help available

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

6.18 MacroCreate

SCPI Command :

```
*DMC
```

class MacroCreateCls

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

get(*label: str*) → str

```
# SCPI: *DMC
value: str = driver.macroCreate.get(label = 'abc')
```

Creates a macro. If the label exists already, the macro contents are overwritten. Macros are deleted when a remote connection is closed but can be saved to a macro file for later reuse, see method RsCmwBase.MassMemory.Store.Macro.set. Avoid using labels which are identical with supported remote control commands. In contrast to SCPI stipulations, remote commands have priority over macros.

param label
No help available

return
macro: No help available

set(*label: str*, *macro: str*) → None

```
# SCPI: *DMC
driver.macroCreate.set(label = 'abc', macro = 'abc')
```

Creates a macro. If the label exists already, the macro contents are overwritten. Macros are deleted when a remote connection is closed but can be saved to a macro file for later reuse, see method RsCmwBase.MassMemory.Store.Macro.set. Avoid using labels which are identical with supported remote control commands. In contrast to SCPI stipulations, remote commands have priority over macros.

param label
No help available

param macro
No help available

6.19 MassMemory

SCPI Commands :

```
MMEMory:COPY
MMEMory:DELete
MMEMory:DRIVes
MMEMory:MDIRectory
MMEMory:MOVE
MMEMory:MSIS
MMEMory:RDIRectory
MMEMory:SAV
MMEMory:RCL
MMEMory:ALIases
```

class MassMemoryCls

MassMemory commands group definition. 22 total commands, 6 Subgroups, 10 group commands

class AliasesStruct

Structure for reading output parameters. Fields:

- Alias: List[str]: No parameter help available
- Path: List[str]: No parameter help available

copy(*file_source: str*, *file_destination: str = None*) → None

```
# SCPI: MMEMory:COPY
driver.massMemory.copy(file_source = 'abc', file_destination = 'abc')
```

Copies an existing file. The target directory must exist.

param file_source
string Name of the file to be copied. Wildcards ? and * are allowed if FileDestination contains a path without filename.

param file_destination
string Path and/or name of the new file If no file destination is specified, the source file is written to the current directory (see method RsCmwBase.MassMemory.CurrentDirectory.set) . Wildcards are not allowed.

delete(*filename: str*) → None

```
# SCPI: MMEMory:DELete
driver.massMemory.delete(filename = 'abc')
```

Deletes the specified files.

param filename
string File to be deleted. The wildcards * and ? are allowed. Specifying a directory instead of a file is not allowed.

delete_directory(*directory_name: str*) → None

```
# SCPI: MMEMory:RDIRectory
driver.massMemory.delete_directory(directory_name = 'abc')
```

Removes an existing empty directory from the mass memory storage system.

param directory_name
string Wildcards are not allowed.

get_aliases() → AliasesStruct

```
# SCPI: MMEMory:ALIases
value: AliasesStruct = driver.massMemory.get_aliases()
```

Returns the defined alias entries and the assigned directories. These settings are predefined and cannot be configured.

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

get_drives() → List[str]

```
# SCPI: MMEMory:DRIVes
value: List[str] = driver.massMemory.get_drives()
```

Returns a list of the available drives.

return
drive: string Comma-separated list of strings, one string per drive

get_msis() → str

```
# SCPI: MMEMory:MSIS
value: str = driver.massMemory.get_msis()
```

Changes the default storage unit (drive or server) for mass memory storage. When the default storage unit is changed, it is checked whether the current directory (see method RsCmwBase.MassMemory.CurrentDirectory.set) is also available on the new storage unit. If not, the current directory is automatically set to '/'.

return
msus: No help available

make_directory(*directory_name: str*) → None

```
# SCPI: MMEMory:MDIRectory
driver.massMemory.make_directory(directory_name = 'abc')
```

Creates a directory. If necessary, an entire path consisting of several subdirectories is created.

param directory_name
string Wildcards are not allowed.

move(*file_source: str*, *file_destination: str*) → None

```
# SCPI: MMEMory:MOVE
driver.massMemory.move(file_source = 'abc', file_destination = 'abc')
```

Moves or renames an existing object (file or directory) to a new location.

param file_source
string Name of the object to be moved or renamed. Wildcards ? and * are only allowed for moving files without renaming.

param file_destination
string New name and/or path of the object. Wildcards are not allowed. If a new object name without path is specified, the object is renamed. If a new path without object name is specified, the object is moved to this path. If a new path and a new object name are specified, the object is moved to this path and renamed.

recall(*filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:RCL
driver.massMemory.recall(filename = 'abc', msus = 'abc')
```

Restores the instrument settings from the specified file. This command has the same effect as the combination of method RsCmwBase.MassMemory.Load.State.set and *RCL.

param filename
No help available

param msus
No help available

save(*filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:SAV
driver.massMemory.save(filename = 'abc', msus = 'abc')
```

Stores the current instrument settings to the specified file. This command has the same effect as the combination of *SAV and method RsCmwBase.MassMemory.Store.State.set.

param filename
No help available

param msus
No help available

set_msis(*msus: str*) → None

```
# SCPI: MMEMory:MSIS
driver.massMemory.set_msis(msus = 'abc')
```

Changes the default storage unit (drive or server) for mass memory storage. When the default storage unit is changed, it is checked whether the current directory (see method RsCmwBase.MassMemory.CurrentDirectory.set) is also available on the new storage unit. If not, the current directory is automatically set to '/'.

param msus
string Default storage unit

Cloning the Group

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

Subgroups

6.19.1 Attribute

SCPI Command :

```
MMEMory:ATTRibute
```

class AttributeCls

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

get(*path_name: str*) → List[str]

```
# SCPI: MMEMory:ATTRibute
value: List[str] = driver.massMemory.attribute.get(path_name = 'abc')
```

Sets or removes file attributes for files and directories.

param path_name
No help available

return
file_entry: No help available

set(*path_name: str*, *attributes: str*) → None

```
# SCPI: MMEMory:ATTRibute
driver.massMemory.attribute.set(path_name = 'abc', attributes = 'abc')
```

Sets or removes file attributes for files and directories.

param path_name
No help available

param attributes
No help available

6.19.2 Catalog

SCPI Command :

```
MMEMory:CATalog
```

class CatalogCls

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

class GetStruct

Response structure. Fields:

- Used_Memory: int: No parameter help available
- Free_Memory: int: No parameter help available
- File_Entry: List[str]: No parameter help available

get(*path_name: str*, *format_py: CatalogFormat = None*) → GetStruct

```
# SCPI: MMEMory:CATalog
value: GetStruct = driver.massMemory.catalog.get(path_name = 'abc', format_py =␣
↪enums.CatalogFormat.ALL)
```

Returns information on the contents of the current or of a specified directory.

param path_name
No help available

param format_py
No help available

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

Cloning the Group

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

Subgroups

6.19.2.1 Length

SCPI Command :

```
MMEMory:CATalog:LENGth
```

class LengthCls

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

get(*path_name: str = None*) → int

```
# SCPI: MMEMory:CATalog:LENGth
value: int = driver.massMemory.catalog.length.get(path_name = 'abc')
```

Returns the number of files and subdirectories of the current or of a specified directory. The number includes the directory strings '.' and '..' so that it corresponds to the number of strings returned by the method RsCmwBase. **MassMemory.Catalog.get_** command after the initial numeric parameters.

param path_name
string If the directory name is omitted, the command queries the contents of the current directory (see method RsCmwBase.MassMemory.CurrentDirectory.set) . If the wildcards ? or * are used, the number of files and subdirectories matching this pattern are returned.

return
count: No help available

6.19.3 CurrentDirectory

SCPI Command :

```
MMEMory:CDIRectory
```

class CurrentDirectoryCls

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

get(*directory_name: str = None*) → str

```
# SCPI: MMEMory:CDIRectory
value: str = driver.massMemory.currentDirectory.get(directory_name = 'abc')
```

Changes the current directory for mass memory storage. If <DirectoryName> is omitted, the current directory is set to '/'.

param directory_name
string Wildcards are not allowed.

return
directory_name: string Wildcards are not allowed.

set(*directory_name: str = None*) → None

```
# SCPI: MMEMory:CDIRectory
driver.massMemory.currentDirectory.set(directory_name = 'abc')
```

Changes the current directory for mass memory storage. If <DirectoryName> is omitted, the current directory is set to '/'.

param directory_name
string Wildcards are not allowed.

6.19.4 Dcatalog

SCPI Command :

```
MMEMory:DCATalog
```

class DcatalogCls

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

get(*path_name: str = None*) → List[str]

```
# SCPI: MMEMory:DCATalog
value: List[str] = driver.massMemory.dcatalog.get(path_name = 'abc')
```

Returns the subdirectories of the current or of a specified directory.

param path_name
No help available

return
file_entry: No help available

Cloning the Group

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

Subgroups

6.19.4.1 Length

SCPI Command :

```
MMEMory:DCATalog:LENGth
```

class LengthCls

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

get(*path_name: str = None*) → int

```
# SCPI: MMEMory:DCATalog:LENGth
value: int = driver.massMemory.dcatalog.length.get(path_name = 'abc')
```

Returns the number of subdirectories of the current or of a specified directory. The number includes the directory strings '.' and '..' so that it corresponds to the number of strings returned by the method RsCmwBase.MassMemory.Dcatalog. **get_** command.

param path_name
string If the directory name is omitted, the command queries the contents of the current directory (see method RsCmwBase.MassMemory.CurrentDirectory.set) . If the wildcards ? or * are used, the number of subdirectories matching this pattern are returned.

return
file_entry_count: No help available

6.19.5 Load

class LoadCls

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

Cloning the Group

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

Subgroups

6.19.5.1 Item

SCPI Command :

```
MMEMory:LOAD:ITEM
```

class ItemCls

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

set(*item_path: str*, *filename: str*) → None

```
# SCPI: MMEMory:LOAD:ITEM
driver.massMemory.load.item.set(item_path = 'abc', filename = 'abc')
```

Executes a partial recall. That means, restores a selected part of a save file. You can restore all settings of a specific application instance. Or you can restore the list mode settings of a specific measurement application instance.

param item_path
No help available

param filename
No help available

6.19.5.2 Macro

SCPI Command :

```
MMEMory:LOAD:MACRo
```

class MacroCls

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

set(*label: str*, *filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:LOAD:MACRo
driver.massMemory.load.macro.set(label = 'abc', filename = 'abc', msus = 'abc')
```

Creates a macro, reading the macro contents from a file. If the label exists already, the macro contents are overwritten. Avoid using labels which are identical with supported remote control commands. In contrast to SCPI stipulations, remote commands have priority over macros.

param label
No help available

param filename
No help available

param msus
No help available

6.19.5.3 State

SCPI Command :

```
MMEMory:LOAD:STATe
```

class StateCls

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

set(*sav_rcl_state_number: float*, *filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:LOAD:STATe
driver.massMemory.load.state.set(sav_rcl_state_number = 1.0, filename = 'abc',␣
↪msus = 'abc')
```

Loads the instrument settings from the specified file to the specified internal memory. After the file has been loaded, the settings must be activated using a *RCL command. For more convenience, see method RsCmwBase.MassMemory.recall.

param sav_rcl_state_number
No help available

param filename
No help available

param msus
No help available

6.19.6 Store

class StoreCls

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

Cloning the Group

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

Subgroups

6.19.6.1 Item

SCPI Command :

```
MMEMory:STORe:ITEM
```

class ItemCls

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

set(*item_name: str*, *filename: str*) → None

```
# SCPI: MMEMory:STORe:ITEM
driver.massMemory.store.item.set(item_name = 'abc', filename = 'abc')
```

Executes a partial save, i.e. stores a part of the instrument settings to the specified file. You can store all settings of a specific application instance. Or you can store the list mode settings of a specific measurement application instance.

param item_name
string Part to be saved. ItemName = Application[i][:MEV:LIST] For Application, see method RsCmwBase.MassMemory.Load.Item.set. i is the instance of the application. Omitting i stores instance 1. Appending :MEV:LIST stores only the list mode settings.

param filename
string Path and filename of the target file. Wildcards are not allowed.

6.19.6.2 Macro

SCPI Command :

```
MMEMory:STORe:MACRo
```

class MacroCls

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

set(*label: str*, *filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:STORe:MACRo
driver.massMemory.store.macro.set(label = 'abc', filename = 'abc', msus = 'abc')
```

Stores the contents of a macro to a file. If the file exists, it is overwritten. If the file does not exist, it is created.

param label
No help available

param filename
No help available

param msus
No help available

6.19.6.3 State

SCPI Command :

```
MMEMory:STORe:STATe
```

class StateCls

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

set(*sav_rcl_state_number: int*, *filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:STORe:STATe
driver.massMemory.store.state.set(sav_rcl_state_number = 1, filename = 'abc',␣
↪msus = 'abc')
```

Stores the instrument settings from the specified internal memory to the specified file. To store the current instrument settings to a file, use first *SAV <MemoryNumber> to store the settings to the memory. Then use this command to store the settings from the memory to a file. For more convenience, see method RsCmwBase.MassMemory.save.

param sav_rcl_state_number
No help available

param filename
No help available

param msus
No help available

6.20 MultiCmw

SCPI Command :

```
INITiate:BASE:MCMW
```

class MultiCmwCls

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

initiate(*cmw_1: CmwSetStatus*, *cmw_2: CmwSetStatus*, *cmw_3: CmwSetStatus*, *cmw_4: CmwSetStatus*) → None

```
# SCPI: INITiate:BASE:MCMW
driver.multiCmw.initiate(cmw_1 = enums.CmwSetStatus.MCMW, cmw_2 = enums.
↪CmwSetStatus.MCMW, cmw_3 = enums.CmwSetStatus.MCMW, cmw_4 = enums.
↪CmwSetStatus.MCMW)
```

Configures the state of CMW 1 to CMW 4 and applies the changes. This command can cause a reboot of the instruments, including firmware updates and typically takes about 10 minutes.

param cmw_1
STBY | SALone | MCMW STBY: standalone mode, standby state SALone: standalone mode, ready state MCMW: multi-CMW mode, ready state

param cmw_2
STBY | SALone | MCMW

param cmw_3
STBY | SALone | MCMW

param cmw_4
STBY | SALone | MCMW

Cloning the Group

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

Subgroups

6.20.1 Identify

SCPI Command :

```
STARt:BASE:MCMW:IDENtify
```

class IdentifyCls

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

start(*box_nr: BoxNumber*, *blinking_time: int = None*) → None

```
# SCPI: STARt:BASE:MCMW:IDENtify
driver.multiCmw.identify.start(box_nr = enums.BoxNumber.BOX1, blinking_time = 1)
```

No command help available

param box_nr
No help available

param blinking_time
No help available

6.20.2 Snumber

SCPI Command :

```
FETCh:BASE:MCMW:SNUMber
```

class SnumberCls

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

fetch(*box_nr: BoxNumber*) → str

```
# SCPI: FETCh:BASE:MCMW:SNUMber
value: str = driver.multiCmw.snumber.fetch(box_nr = enums.BoxNumber.BOX1)
```

No command help available

param box_nr
No help available

return
serial_number: No help available

6.20.3 State

SCPI Command :

```
FETCh:BASE:MCMW:STATe
```

class StateCls

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

class FetchStruct

Response structure. Fields:

- Cmw_1: enums.CmwCurrentStatus: STBY | SALone | MCMW | ERRor | MCCNconnected | PCINconnected STBY: standalone mode, standby state SALone: standalone mode, ready state MCMW: multi-CMW mode, ready state ERRor: an error occurred, a state change failed MCCNconnected: there is no MCC connection to the instrument, so the current state cannot be queried or changed PCINconnected: there is no PCIe connection to the instrument
- Cmw_2: enums.CmwCurrentStatus: STBY | SALone | MCMW | ERRor | MCCNconnected | PCINconnected
- Cmw_3: enums.CmwCurrentStatus: STBY | SALone | MCMW | ERRor | MCCNconnected | PCINconnected
- Cmw_4: enums.CmwCurrentStatus: STBY | SALone | MCMW | ERRor | MCCNconnected | PCINconnected

fetch() → FetchStruct

```
# SCPI: FETCh:BASE:MCMW:STATe
value: FetchStruct = driver.multiCmw.state.fetch()
```

Queries the current state of CMW 1 to CMW 4.

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

6.21 Procedure

SCPI Command :

```
PROCedure:CMWD
```

class ProcedureCls

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

get_cmwd() → str

```
# SCPI: PROCedure:CMWD
value: str = driver.procedure.get_cmwd()
```

No command help available

return
command_string: No help available

set_cmwd(*command_string: str*) → None

```
# SCPI: PROCedure:CMWD
driver.procedure.set_cmwd(command_string = 'abc')
```

No command help available

param command_string
No help available

6.22 RecallState

SCPI Command :

```
*RCL
```

class RecallStateCls

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

set(*num: float*) → None

```
# SCPI: *RCL
driver.recallState.set(num = 1.0)
```

Loads the instrument settings from an intermediate memory identified by the specified number. The instrument settings can be stored to this memory using the command *SAV with the associated number. To load instrument settings from a file to the memory, see method RsCmwBase.MassMemory.Load.State.set. See also method RsCmwBase.MassMemory.recall.

param num
integer Range: 0 to 99

6.23 SaveState

SCPI Command :

```
*SAV
```

class SaveStateCls

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

set(*num: float*) → None

```
# SCPI: *SAV
driver.saveState.set(num = 1.0)
```

Stores the current instrument settings under the specified number in an intermediate memory. The settings can be restored, using the command *RCL with the associated number. To save the stored instrument settings to a file, see method RsCmwBase. MassMemory.Store.State.set. See also method RsCmwBase.MassMemory.save.

param num
integer Range: 0 to 99

6.24 Sense

class SenseCls

Sense commands group definition. 13 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.24.1 FirmwareUpdate

SCPI Command :

```
SENSe:FWUPdate:INFO
```

class FirmwareUpdateCls

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

get_info() → str

```
# SCPI: SENSe:FWUPdate:INFO
value: str = driver.sense.firmwareUpdate.get_info()
```

No command help available

return
info: No help available

6.24.2 IpSet

class IpSetCls

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

Cloning the Group

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

Subgroups

6.24.2.1 Snode

SCPI Commands :

```
SENSe:BASE:IPSet:SNODe:NNAMe
SENSe:BASE:IPSet:SNODe:NTYPe
SENSe:BASE:IPSet:SNODe:NSEGment
```

class SnodeCls

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

class NsegmentStruct

Structure for reading output parameters. Fields:

- Selected_Segment: enums.Segment: A | B | C Selected network segment
- Ip_Address: str: string First two IP address octets of the subnet
- Subnet_Mask: str: string Used subnet mask (fixed value)

get_nname() → str

```
# SCPI: SENSe:BASE:IPSet:SNODe:NNAMe
value: str = driver.sense.ipSet.snode.get_nname()
```

Queries the subnet node name of the R&S CMW.

return
name: string

get_nsegment() → NsegmentStruct

```
# SCPI: SENSe:BASE:IPSet:SNODe:NSEGment
value: NsegmentStruct = driver.sense.ipSet.snode.get_nsegment()
```

Queries information about the selected network segment and the resulting subnet properties.

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

get_ntype() → str

```
# SCPI: SENSe:BASE:IPSet:SNODe:NTYPe
value: str = driver.sense.ipSet.snode.get_ntype()
```

Queries the subnet node type of the R&S CMW.

return
type_py: string 'CMW'

6.24.2.2 SubMonitor

SCPI Commands :

```
SENSe:BASE:IPSet:SMONitor:NAME
SENSe:BASE:IPSet:SMONitor:TYPE
SENSe:BASE:IPSet:SMONitor:ID
SENSe:BASE:IPSet:SMONitor:DESCription
```

class SubMonitorCls

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

get_description() → List[str]

```
# SCPI: SENSe:BASE:IPSet:SMONitor:DESCription
value: List[str] = driver.sense.ipSet.subMonitor.get_description()
```

Queries the description of all network nodes detected by the subnet monitor.

return
descriptions: string Comma-separated list of strings, one per network node

get_id() → List[int]

```
# SCPI: SENSe:BASE:IPSet:SMONitor:ID
value: List[int] = driver.sense.ipSet.subMonitor.get_id()
```

Queries the ID (third segment of IP address) of all network nodes detected by the subnet monitor.

return
ids: decimal Comma-separated list of values, one per network node Range: 1 to 254

get_name() → List[str]

```
# SCPI: SENSe:BASE:IPSet:SMONitor:NAME
value: List[str] = driver.sense.ipSet.subMonitor.get_name()
```

Queries the name of all network nodes detected by the subnet monitor.

return
names: string Comma-separated list of strings, one per network node

get_type_py() → List[str]

```
# SCPI: SENSe:BASE:IPSet:SMONitor:TYPE
value: List[str] = driver.sense.ipSet.subMonitor.get_type_py()
```

Queries the type of all network nodes detected by the subnet monitor.

return
types: string Comma-separated list of strings, one per network node

6.24.3 Reference

class ReferenceCls

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

Cloning the Group

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

Subgroups

6.24.3.1 Frequency

SCPI Command :

```
SENSe:BASE:REFerence:FREQuency:LOCKed
```

class FrequencyCls

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

get_locked() → bool

```
# SCPI: SENSe:BASE:REFerence:FREQuency:LOCKed
value: bool = driver.sense.reference.frequency.get_locked()
```

Queries whether the reference frequency is locked or not.

return
lock: 1 | 0 1: The frequency is locked. 0: The frequency is not locked.

6.24.4 Temperature

SCPI Command :

```
SENSe:BASE:TEMPerature:ENVironment
```

class TemperatureCls

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

get_environment() → float

```
# SCPI: SENSe:BASE:TEMPerature:ENVironment
value: float = driver.sense.temperature.get_environment()
```

No command help available

return
temperature: No help available

Cloning the Group

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

Subgroups

6.24.4.1 Exceeded

SCPI Commands :

```
SENSe:BASE:TEMPerature:EXCeeded:LIST
SENSe:BASE:TEMPerature:EXCeeded
```

class ExceededCls

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

class ListPyStruct

Structure for reading output parameters. Fields:

- Meas_Point: List[str]: No parameter help available
- Current_Temp: List[float]: No parameter help available
- Max_Temp: List[float]: No parameter help available

get_list_py() → ListPyStruct

```
# SCPI: SENSe:BASE:TEMPerature:EXCeeded:LIST
value: ListPyStruct = driver.sense.temperature.exceeded.get_list_py()
```

No command help available

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

get_value() → bool

```
# SCPI: SENSe:BASE:TEMPerature:EXCeeded
value: bool = driver.sense.temperature.exceeded.get_value()
```

No command help available

return
exceed: No help available

6.24.4.2 Operating

SCPI Command :

```
SENSe:BASE:TEMPerature:OPERating:INTernal
```

class OperatingCls

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

get_internal() → float

```
# SCPI: SENSe:BASE:TEMPerature:OPERating:INTernal
value: float = driver.sense.temperature.operating.get_internal()
```

Queries the temperature within the instrument. The returned value indicates the average of the temperatures measured at the individual RF modules. The recommended temperature range is illustrated in the following figure.

return
temperature: float Temperature in degrees Unit: °C

6.25 Source

class SourceCls

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

Cloning the Group

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

Subgroups

6.25.1 Adjustment

class AdjustmentCls

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

Cloning the Group

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

Subgroups

6.25.1.1 State

SCPI Command :

```
SOURce:BASE:ADJustment:STATe
```

class StateCls

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

get() → BaseAdjState

```
# SCPI: SOURce:BASE:ADJustment:STATe
value: enums.BaseAdjState = driver.source.adjustment.state.get()
```

No command help available

return
state: No help available

set(*control: bool*) → None

```
# SCPI: SOURce:BASE:ADJustment:STATe
driver.source.adjustment.state.set(control = False)
```

No command help available

param control
No help available

6.26 Status

SCPI Command :

```
STATus:PRESet
```

class StatusCls

Status commands group definition. 37 total commands, 7 Subgroups, 1 group commands

preset() → None

```
# SCPI: STATus:PRESet
driver.status.preset()
```

Configures the status reporting system such that device-dependent events are not reported at a higher level.
INTRO_CMD_HELP: The command affects only the transition filter registers, the ENABle registers, and queue enabling:

- The ENABle parts of the STATus:OPERation and STATus:QUEStionable… registers are set to all 0's.

- The PTRansition parts are set all 1's, the NTRansition parts are set to all 0's, so that only positive transitions in the CONDition part are recognized.

The status reporting system is also affected by other commands, see 'Reset values of the status reporting system'.

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

```
# SCPI: STATus:PRESet
driver.status.preset_with_opc()
```

Configures the status reporting system such that device-dependent events are not reported at a higher level.

INTRO_CMD_HELP: The command affects only the transition filter registers, the ENABle registers, and queue enabling:

- The ENABle parts of the STATus:OPERation and STATus:QUEStionable… registers are set to all 0's.
- The PTRansition parts are set all 1's, the NTRansition parts are set to all 0's, so that only positive transitions in the CONDition part are recognized.

The status reporting system is also affected by other commands, see 'Reset values of the status reporting system'.

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

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

Cloning the Group

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

Subgroups

6.26.1 Condition

class ConditionCls

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

Cloning the Group

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

Subgroups

6.26.1.1 Bits

class BitsCls

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

Cloning the Group

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

Subgroups

6.26.1.1.1 All

SCPI Command :

```
STATus:CONDition:BITS:ALL
```

class AllCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → List[str]

```
# SCPI: STATus:CONDition:BITS:ALL
value: List[str] = driver.status.condition.bits.all.get(filter_py = 'abc', mode␣
↪= enums.ExpressionMode.REGex)
```

This command offers a comfortable way to get an overview of all task states, without querying each register individually. It evaluates the CONDition parts of the lowest level OPERation status registers. The command is nondestructive. In most situations, the returned list shows all task states of the installed firmware applications. However it can happen that a task is not listed if currently no resources at all are assigned to that task (e.g. directly after installation) . In that case, you could say that the state of the task is less than 'OFF'.

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.26.1.1.2 Cataloge

SCPI Command :

```
STATus:CONDition:BITS:CATaloge
```

class CatalogeCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → List[str]

```
# SCPI: STATus:CONDition:BITS:CATaloge
value: List[str] = driver.status.condition.bits.cataloge.get(filter_py = 'abc',␣
↪mode = enums.ExpressionMode.REGex)
```

Returns a list of all possible task states for the installed firmware applications. The current task states returned by method **RsCmwBase.Status.Condition.Bits.All.get_** form a subset of the list returned by this command.

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.26.1.1.3 Count

SCPI Command :

```
STATus:CONDition:BITS:COUNt
```

class CountCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → int

```
# SCPI: STATus:CONDition:BITS:COUNt
value: int = driver.status.condition.bits.count.get(filter_py = 'abc', mode =␣
↪enums.ExpressionMode.REGex)
```

Returns the number of task states listed by method **RsCmwBase.Status.Condition.Bits.All.get_**.

param filter_py
No help available

param mode
No help available

return
count: decimal

6.26.2 Event

class EventCls

Event commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.26.2.1 Bits

SCPI Command :

```
STATus:EVENt:BITS:CLEar
```

class BitsCls

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

clear(*filter_py: str = None*, *mode: ExpressionMode = None*) → None

```
# SCPI: STATus:EVENt:BITS:CLEar
driver.status.event.bits.clear(filter_py = 'abc', mode = enums.ExpressionMode.
↪REGex)
```

Clears the EVENt parts of all status registers of the STATus:OPERation register hierarchy. If a regular expression is defined, the command is only applied to the registers matching the filter criteria.

param filter_py
No help available

param mode
No help available

Cloning the Group

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

Subgroups

6.26.2.1.1 All

SCPI Command :

```
STATus:EVENt:BITS:ALL
```

class AllCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → List[str]

```
# SCPI: STATus:EVENt:BITS:ALL
value: List[str] = driver.status.event.bits.all.get(filter_py = 'abc', mode =␣
↪enums.ExpressionMode.REGex)
```

Evaluates the EVENt parts of all lowest level OPERation status registers. The command is nondestructive. This command offers a comfortable way to get an overview of the EVENt parts of all lowest level registers, without querying each register individually.

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.26.2.1.2 Count

SCPI Command :

```
STATus:EVENt:BITS:COUNt
```

class CountCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → int

```
# SCPI: STATus:EVENt:BITS:COUNt
value: int = driver.status.event.bits.count.get(filter_py = 'abc', mode = enums.
↪ExpressionMode.REGex)
```

Returns the number of events listed by method **RsCmwBase.Status.Event.Bits.All.get_**.

param filter_py
No help available

param mode
No help available

return
count: decimal

6.26.2.1.3 Next

SCPI Command :

```
STATus:EVENt:BITS:NEXT
```

class NextCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:EVENt:BITS:NEXT
value: str = driver.status.event.bits.next.get(filter_py = 'abc', mode = enums.
↪ExpressionMode.REGex)
```

Searches, returns and deletes the next event at the lowest level of the STATus:OPERation register hierarchy. This command can be used to supply state transitions to a remote control program one by one. The program can then react on the transitions, e.g. fetch the results of a measurement that reached the RDY or SDR state. Or start a new measurement after a measurement has been finished. A list of all events in the STATus:OPERation register hierarchy can be returned using method **RsCmwBase.Status.Event.Bits.All.get_**.

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.26.3 Generator

class GeneratorCls

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

Cloning the Group

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

Subgroups

6.26.3.1 Condition

class ConditionCls

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

Cloning the Group

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

Subgroups

6.26.3.1.1 Off

SCPI Command :

```
STATus:GENerator:CONDition:OFF
```

class OffCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:GENerator:CONDition:OFF
value: str = driver.status.generator.condition.off.get(filter_py = 'abc', mode
↪= enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.3.1.2 On

SCPI Command :

```
STATus:GENerator:CONDition:ON
```

class OnCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:GENerator:CONDition:ON
value: str = driver.status.generator.condition.on.get(filter_py = 'abc', mode =
↪enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.3.1.3 Pending

SCPI Command :

```
STATus:GENerator:CONDition:PENDing
```

class PendingCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:GENerator:CONDition:PENDing
value: str = driver.status.generator.condition.pending.get(filter_py = 'abc',␣
↪mode = enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.4 Measurement

class MeasurementCls

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

Cloning the Group

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

Subgroups

6.26.4.1 Condition

class ConditionCls

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

Cloning the Group

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

Subgroups

6.26.4.1.1 Off

SCPI Command :

```
STATus:MEASurement:CONDition:OFF
```

class OffCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:OFF
value: str = driver.status.measurement.condition.off.get(filter_py = 'abc',␣
↪mode = enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.4.1.2 Qued

SCPI Command :

```
STATus:MEASurement:CONDition:QUED
```

class QuedCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:QUED
value: str = driver.status.measurement.condition.qued.get(filter_py = 'abc',␣
↪mode = enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.4.1.3 Rdy

SCPI Command :

```
STATus:MEASurement:CONDition:RDY
```

class RdyCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:RDY
value: str = driver.status.measurement.condition.rdy.get(filter_py = 'abc',␣
↪mode = enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.4.1.4 Run

SCPI Command :

```
STATus:MEASurement:CONDition:RUN
```

class RunCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:RUN
value: str = driver.status.measurement.condition.run.get(filter_py = 'abc',␣
↪mode = enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.4.1.5 SdReached

SCPI Command :

```
STATus:MEASurement:CONDition:SDReached
```

class SdReachedCls

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

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:SDReached
value: str = driver.status.measurement.condition.sdReached.get(filter_py = 'abc
↪', mode = enums.ExpressionMode.REGex)
```

Lists all generator tasks or measurement tasks whose current state equals the state indicated by the last mnemonic. The results are collected from the CONDition parts of the lowest level registers of the STATus:OPERation register hierarchy.

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.26.5 Operation

SCPI Commands :

```
STATus:OPERation[:EVENt]
STATus:OPERation:CONDition
STATus:OPERation:ENABle
STATus:OPERation:PTRansition
STATus:OPERation:NTRansition
```

class OperationCls

Operation commands group definition. 10 total commands, 1 Subgroups, 5 group commands

get_condition() → int

```
# SCPI: STATus:OPERation:CONDition
value: int = driver.status.operation.get_condition()
```

Returns the contents of the CONDition part of the status register. Reading the CONDition registers is nondestructive. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

return
register_value: integer Range: 0 to 65535 (decimal representation)

get_enable() → int

```
# SCPI: STATus:OPERation:ENABle
value: int = driver.status.operation.get_enable()
```

Sets the enable mask which allows true conditions in the EVENt part of the status register to be reported to the next higher level in the summary bit. If a bit is 1 in the enable register and the associated event bit changes to true, a positive transition occurs in the summary bit. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

return
register_value: No help available

get_event() → int

```
# SCPI: STATus:OPERation[:EVENt]
value: int = driver.status.operation.get_event()
```

Returns the contents of the EVENt part of the status register. Reading an EVENt part clears it. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

return
register_value: integer Range: 0 to 65535 (decimal representation)

get_ntransition() → int

```
# SCPI: STATus:OPERation:NTRansition
value: int = driver.status.operation.get_ntransition()
```

Sets the negative transition filter. If a bit is set, a 1 to 0 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

return
register_value: No help available

get_ptransition() → int

```
# SCPI: STATus:OPERation:PTRansition
value: int = driver.status.operation.get_ptransition()
```

Sets the positive transition filter. If a bit is set, a 0 to 1 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

return
register_value: No help available

set_enable(*register_value: int*) → None

```
# SCPI: STATus:OPERation:ENABle
driver.status.operation.set_enable(register_value = 1)
```

Sets the enable mask which allows true conditions in the EVENt part of the status register to be reported to the next higher level in the summary bit. If a bit is 1 in the enable register and the associated event bit changes to true, a positive transition occurs in the summary bit. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

param register_value
integer Range: 0 to 65535 (decimal representation)

set_ntransition(*register_value: int*) → None

```
# SCPI: STATus:OPERation:NTRansition
driver.status.operation.set_ntransition(register_value = 1)
```

Sets the negative transition filter. If a bit is set, a 1 to 0 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

param register_value
integer Range: 0 to 65535 (decimal representation)

set_ptransition(*register_value: int*) → None

```
# SCPI: STATus:OPERation:PTRansition
driver.status.operation.set_ptransition(register_value = 1)
```

Sets the positive transition filter. If a bit is set, a 0 to 1 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register. For a description of the variables <netw_std>, <func_grp> and <appl>, refer to Table 'Variables in STATus:OPERation commands'.

param register_value
integer Range: 0 to 65535 (decimal representation)

Cloning the Group

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

Subgroups

6.26.5.1 Bit<BitNr>

RepCap Settings

```
# Range: Nr8 .. Nr12
rc = driver.status.operation.bit.repcap_bitNr_get()
driver.status.operation.bit.repcap_bitNr_set(repcap.BitNr.Nr8)
```

class BitCls

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: BitNr, default value after init: BitNr.Nr8

Cloning the Group

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

Subgroups

6.26.5.1.1 Condition

SCPI Command :

```
STATus:OPERation:BIT<bitno>:CONDition
```

class ConditionCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>:CONDition
value: bool = driver.status.operation.bit.condition.get(bitNr = repcap.BitNr.
↪Default)
```

Returns bit no. <n> of the CONDition or EVENt part of the STATus:OPERation register. To return the entire parts, see method RsCmwBase.Status.Operation.condition and method RsCmwBase.Status.Operation.event.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.26.5.1.2 Enable

SCPI Command :

```
STATus:OPERation:BIT<bitno>:ENABle
```

class EnableCls

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

get(*bitNr=BitNr.Default*) → float

```
# SCPI: STATus:OPERation:BIT<bitno>:ENABle
value: float = driver.status.operation.bit.enable.get(bitNr = repcap.BitNr.
↪Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:OPERation register. To set the entire parts, see method RsCmwBase.Status.Operation.enable, method RsCmwBase.Status.Operation.ntransition and method RsCmwBase.Status. Operation.ptransition.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: float*, *bitNr=BitNr.Default*) → None

```
# SCPI: STATus:OPERation:BIT<bitno>:ENABle
driver.status.operation.bit.enable.set(register_bit = 1.0, bitNr = repcap.BitNr.
↪Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:OPERation register. To set the entire parts, see method RsCmwBase.Status.Operation.enable, method RsCmwBase.Status.Operation.ntransition and method RsCmwBase.Status. Operation.ptransition.

param register_bit
No help available

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.26.5.1.3 Event

SCPI Command :

```
STATus:OPERation:BIT<bitno>[:EVENt]
```

class EventCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>[:EVENt]
value: bool = driver.status.operation.bit.event.get(bitNr = repcap.BitNr.
↪Default)
```

Returns bit no. <n> of the CONDition or EVENt part of the STATus:OPERation register. To return the entire parts, see method RsCmwBase.Status.Operation.condition and method RsCmwBase.Status.Operation.event.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.26.5.1.4 Ntransition

SCPI Command :

```
STATus:OPERation:BIT<bitno>:NTRansition
```

class NtransitionCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>:NTRansition
value: bool = driver.status.operation.bit.ntransition.get(bitNr = repcap.BitNr.
↪Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:OPERation register. To set the entire parts, see method RsCmwBase.Status.Operation.enable, method RsCmwBase.Status.Operation.ntransition and method RsCmwBase.Status. Operation.ptransition.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bitNr=BitNr.Default*) → None

```
# SCPI: STATus:OPERation:BIT<bitno>:NTRansition
driver.status.operation.bit.ntransition.set(register_bit = False, bitNr =␣
↪repcap.BitNr.Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:OPERation register. To set the entire parts, see method RsCmwBase.Status.Operation.enable, method RsCmwBase.Status.Operation.ntransition and method RsCmwBase.Status. Operation.ptransition.

param register_bit
No help available

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.26.5.1.5 Ptransition

SCPI Command :

```
STATus:OPERation:BIT<bitno>:PTRansition
```

class PtransitionCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>:PTRansition
value: bool = driver.status.operation.bit.ptransition.get(bitNr = repcap.BitNr.
↪Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:OPERation register. To set the entire parts, see method RsCmwBase.Status.Operation.enable, method RsCmwBase.Status.Operation.ntransition and method RsCmwBase.Status. Operation.ptransition.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bitNr=BitNr.Default*) → None

```
# SCPI: STATus:OPERation:BIT<bitno>:PTRansition
driver.status.operation.bit.ptransition.set(register_bit = False, bitNr =␣
↪repcap.BitNr.Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:OPERation register. To set the entire parts, see method RsCmwBase.Status.Operation.enable, method RsCmwBase.Status.Operation.ntransition and method RsCmwBase.Status. Operation.ptransition.

param register_bit
No help available

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.26.6 Questionable

SCPI Commands :

```
STATus:QUEStionable[:EVENt]
STATus:QUEStionable:CONDition
STATus:QUEStionable:ENABle
STATus:QUEStionable:PTRansition
STATus:QUEStionable:NTRansition
```

class QuestionableCls

Questionable commands group definition. 10 total commands, 1 Subgroups, 5 group commands

get_condition() → int

```
# SCPI: STATus:QUEStionable:CONDition
value: int = driver.status.questionable.get_condition()
```

Returns the contents of the CONDition part of the status register. Reading the CONDition registers is nondestructive.

return
register_value: integer Range: 0 to 65535 (decimal representation)

get_enable() → int

```
# SCPI: STATus:QUEStionable:ENABle
value: int = driver.status.questionable.get_enable()
```

Sets the enable mask which allows true conditions in the EVENt part of the status register to be reported to the next higher level in the summary bit. If a bit is 1 in the enable register and its associated event bit changes to true, a positive transition occurs in the summary bit.

return
register_value: No help available

get_event() → int

```
# SCPI: STATus:QUEStionable[:EVENt]
value: int = driver.status.questionable.get_event()
```

Returns the contents of the EVENt part of the status register. Reading an EVENt register clears it.

return
register_value: integer Range: 0 to 65535 (decimal representation)

get_ntransition() → int

```
# SCPI: STATus:QUEStionable:NTRansition
value: int = driver.status.questionable.get_ntransition()
```

Sets the negative transition filter. If a bit is set, a 1 to 0 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register.

return
register_value: No help available

get_ptransition() → int

```
# SCPI: STATus:QUEStionable:PTRansition
value: int = driver.status.questionable.get_ptransition()
```

Sets the positive transition filter. If a bit is set, a 0 to 1 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register.

return
register_value: No help available

set_enable(*register_value: int*) → None

```
# SCPI: STATus:QUEStionable:ENABle
driver.status.questionable.set_enable(register_value = 1)
```

Sets the enable mask which allows true conditions in the EVENt part of the status register to be reported to the next higher level in the summary bit. If a bit is 1 in the enable register and its associated event bit changes to true, a positive transition occurs in the summary bit.

param register_value
integer Range: 0 to 65535 (decimal representation)

set_ntransition(*register_value: int*) → None

```
# SCPI: STATus:QUEStionable:NTRansition
driver.status.questionable.set_ntransition(register_value = 1)
```

Sets the negative transition filter. If a bit is set, a 1 to 0 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register.

param register_value
integer Range: 0 to 65535 (decimal representation)

set_ptransition(*register_value: int*) → None

```
# SCPI: STATus:QUEStionable:PTRansition
driver.status.questionable.set_ptransition(register_value = 1)
```

Sets the positive transition filter. If a bit is set, a 0 to 1 transition in the corresponding bit of the condition register writes a 1 to the corresponding bit of the event register.

param register_value
integer Range: 0 to 65535 (decimal representation)

Cloning the Group

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

Subgroups

6.26.6.1 Bit<BitNr>

RepCap Settings

```
# Range: Nr8 .. Nr12
rc = driver.status.questionable.bit.repcap_bitNr_get()
driver.status.questionable.bit.repcap_bitNr_set(repcap.BitNr.Nr8)
```

class BitCls

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: BitNr, default value after init: BitNr.Nr8

Cloning the Group

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

Subgroups

6.26.6.1.1 Condition

SCPI Command :

```
STATus:QUEStionable:BIT<bitno>:CONDition
```

class ConditionCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:CONDition
value: bool = driver.status.questionable.bit.condition.get(bitNr = repcap.BitNr.
↪Default)
```

Returns bit no. <n> of the CONDition or EVENt part of the STATus:QUEStionable register. To return the entire parts, see method RsCmwBase.Status.Questionable.condition and method RsCmwBase.Status.Questionable.event.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.26.6.1.2 Enable

SCPI Command :

```
STATus:QUEStionable:BIT<bitno>:ENABle
```

class EnableCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:ENABle
value: bool = driver.status.questionable.bit.enable.get(bitNr = repcap.BitNr.
↪Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:QUEStionable register. To set the entire parts, see method RsCmwBase.Status.Questionable.enable, method RsCmwBase.Status.Questionable.ntransition and method RsCmwBase.Status.Questionable.ptransition.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bitNr=BitNr.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<bitno>:ENABle
driver.status.questionable.bit.enable.set(register_bit = False, bitNr = repcap.
↪BitNr.Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:QUEStionable register. To set the entire parts, see method RsCmwBase.Status.Questionable.enable, method RsCmwBase.Status.Questionable.ntransition and method RsCmwBase.Status.Questionable.ptransition.

param register_bit
No help available

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.26.6.1.3 Event

SCPI Command :

```
STATus:QUEStionable:BIT<bitno>[:EVENt]
```

class EventCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>[:EVENt]
value: bool = driver.status.questionable.bit.event.get(bitNr = repcap.BitNr.
↪Default)
```

Returns bit no. <n> of the CONDition or EVENt part of the STATus:QUEStionable register. To return the entire parts, see method RsCmwBase.Status.Questionable.condition and method RsCmwBase.Status.Questionable.event.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.26.6.1.4 Ntransition

SCPI Command :

```
STATus:QUEStionable:BIT<bitno>:NTRansition
```

class NtransitionCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:NTRansition
value: bool = driver.status.questionable.bit.ntransition.get(bitNr = repcap.
↪BitNr.Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:QUEStionable register. To set the entire parts, see method RsCmwBase.Status.Questionable.enable, method RsCmwBase.Status.Questionable.ntransition and method RsCmwBase.Status.Questionable.ptransition.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bitNr=BitNr.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<bitno>:NTRansition
driver.status.questionable.bit.ntransition.set(register_bit = False, bitNr =␣
↪repcap.BitNr.Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:QUEStionable register. To set the entire parts, see method RsCmwBase.Status.Questionable.enable, method RsCmwBase.Status.Questionable.ntransition and method RsCmwBase.Status.Questionable.ptransition.

param register_bit
No help available

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.26.6.1.5 Ptransition

SCPI Command :

```
STATus:QUEStionable:BIT<bitno>:PTRansition
```

class PtransitionCls

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

get(*bitNr=BitNr.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:PTRansition
value: bool = driver.status.questionable.bit.ptransition.get(bitNr = repcap.
↪BitNr.Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:QUEStionable register. To set the entire parts, see method RsCmwBase.Status.Questionable.enable, method RsCmwBase.Status.Questionable.ntransition and method RsCmwBase.Status.Questionable.ptransition.

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bitNr=BitNr.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<bitno>:PTRansition
driver.status.questionable.bit.ptransition.set(register_bit = False, bitNr =␣
↪repcap.BitNr.Default)
```

Sets bit no. <n> of the ENABle, NTRansition or PTRansition part of the STATus:QUEStionable register. To set the entire parts, see method RsCmwBase.Status.Questionable.enable, method RsCmwBase.Status.Questionable.ntransition and method RsCmwBase.Status.Questionable.ptransition.

param register_bit
No help available

param bitNr
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.26.7 Queue

SCPI Command :

```
STATus:QUEue[:NEXT]
```

class QueueCls

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

class NextStruct

Structure for reading output parameters. Fields:

- Error_Code: int: No parameter help available
- Error_Description: str: No parameter help available

get_next() → NextStruct

```
# SCPI: STATus:QUEue[:NEXT]
value: NextStruct = driver.status.queue.get_next()
```

Queries and at the same time deletes the oldest entry in the error queue. Operation is identical to that of SYSTem:ERRor[:NEXT]?.

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

6.27 System

SCPI Commands :

```
SYSTem:BASE:RELiability
SYSTem:DID
SYSTem:KLOCk
SYSTem:PRESet
SYSTem:PRESet:ALL
SYSTem:PRESet:BASE
```

(continues on next page)

(continued from previous page)

```
SYSTem:RESet
SYSTem:RESet:ALL
SYSTem:RESet:BASE
SYSTem:VERSion
```

class SystemCls

System commands group definition. 98 total commands, 25 Subgroups, 10 group commands

get_did() → str

```
# SCPI: SYSTem:DID
value: str = driver.system.get_did()
```

No command help available

return
device_id: No help available

get_klock() → bool

```
# SCPI: SYSTem:KLOCk
value: bool = driver.system.get_klock()
```

Locks or unlocks the local controls of the instrument, including the (soft-) front panel keys.

return
klock: No help available

get_reliability() → int

```
# SCPI: SYSTem:BASE:RELiability
value: int = driver.system.get_reliability()
```

Returns a reliability value indicating errors detected by the base software.

return
value: decimal For reliability indicator values, see 'Checking the reliability indicator'

get_version() → float

```
# SCPI: SYSTem:VERSion
value: float = driver.system.get_version()
```

Queries the SCPI version number to which the instrument complies.

return
version: string '1999.0' is the final SCPI version.

preset(*appl_name_and_li_number: str = None*) → None

```
# SCPI: SYSTem:PRESet
driver.system.preset(appl_name_and_li_number = 'abc')
```

A PRESet sets the parameters of the subinstrument to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation. Optionally, the preset/reset can be limited to a specific application instance.

param appl_name_and_li_number
string Application and instance to be reset/preset. Example: 'LTE Meas1' for LTE UE measurements instance 1 Omitting the instance (e.g. 'LTE Meas') selects instance 1. The supported strings are listed in the table below.

preset_all() → None

```
# SCPI: SYSTem:PRESet:ALL
driver.system.preset_all()
```

A PRESet sets the parameters of all subinstruments and the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

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

```
# SCPI: SYSTem:PRESet:ALL
driver.system.preset_all_with_opc()
```

A PRESet sets the parameters of all subinstruments and the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

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

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

preset_base() → None

```
# SCPI: SYSTem:PRESet:BASE
driver.system.preset_base()
```

A PRESet sets the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

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

```
# SCPI: SYSTem:PRESet:BASE
driver.system.preset_base_with_opc()
```

A PRESet sets the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

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

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

reset(*appl_name_and_li_number: str = None*) → None

```
# SCPI: SYSTem:RESet
driver.system.reset(appl_name_and_li_number = 'abc')
```

A PRESet sets the parameters of the subinstrument to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation. Optionally, the preset/reset can be limited to a specific application instance.

param appl_name_and_li_number
string Application and instance to be reset/preset. Example: 'LTE Meas1' for LTE UE measurements instance 1 Omitting the instance (e.g. 'LTE Meas') selects instance 1. The supported strings are listed in the table below.

reset_all() → None

```
# SCPI: SYSTem:RESet:ALL
driver.system.reset_all()
```

A PRESet sets the parameters of all subinstruments and the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

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

```
# SCPI: SYSTem:RESet:ALL
driver.system.reset_all_with_opc()
```

A PRESet sets the parameters of all subinstruments and the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

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

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

reset_base() → None

```
# SCPI: SYSTem:RESet:BASE
driver.system.reset_base()
```

A PRESet sets the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

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

```
# SCPI: SYSTem:RESet:BASE
driver.system.reset_base_with_opc()
```

A PRESet sets the base settings to default values suitable for local/manual interaction. A RESet sets them to default values suitable for remote operation.

Same as reset_base, but waits for the operation to complete before continuing further. Use the RsCmwBase.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_klock(*klock: bool*) → None

```
# SCPI: SYSTem:KLOCk
driver.system.set_klock(klock = False)
```

Locks or unlocks the local controls of the instrument, including the (soft-) front panel keys.

param klock
ON | OFF | 1 | 0 ON | 1: Local key locked (key lock enabled) OFF | 0: Local keys unlocked

Cloning the Group

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

Subgroups

6.27.1 Cmw<CmwVariant>

RepCap Settings

```
# Range: Cmw1 .. Cmw100
rc = driver.system.cmw.repcap_cmwVariant_get()
driver.system.cmw.repcap_cmwVariant_set(repcap.CmwVariant.Cmw1)
```

class CmwCls

Cmw commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: CmwVariant, default value after init: CmwVariant.Cmw1

Cloning the Group

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

Subgroups

6.27.1.1 Device

class DeviceCls

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

Cloning the Group

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

Subgroups

6.27.1.1.1 Id

SCPI Command :

```
SYSTem:CMW<n>:DEVice:ID
```

class IdCls

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

get(*cmwVariant=CmwVariant.Default*) → str

```
# SCPI: SYSTem:CMW<n>:DEVice:ID
value: str = driver.system.cmw.device.id.get(cmwVariant = repcap.CmwVariant.
↪Default)
```

No command help available

param cmwVariant
optional repeated capability selector. Default value: Cmw1 (settable in the interface 'Cmw')

return
idn: No help available

6.27.2 Communicate

class CommunicateCls

Communicate commands group definition. 19 total commands, 7 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.27.2.1 Gpib<GpibInstance>

RepCap Settings

```
# Range: Inst1 .. Inst32
rc = driver.system.communicate.gpib.repcap_gpibInstance_get()
driver.system.communicate.gpib.repcap_gpibInstance_set(repcap.GpibInstance.Inst1)
```

class GpibCls

Gpib commands group definition. 3 total commands, 2 Subgroups, 0 group commands Repeated Capability: GpibInstance, default value after init: GpibInstance.Inst1

Cloning the Group

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

Subgroups

6.27.2.1.1 Self

class SelfCls

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

Cloning the Group

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

Subgroups

6.27.2.1.1.1 Addr

SCPI Command :

```
SYSTem:COMMunicate:GPIB<inst>[:SELF]:ADDR
```

class AddrCls

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

get(*gpibInstance=GpibInstance.Default*) → int

```
# SCPI: SYSTem:COMMunicate:GPIB<inst>[:SELF]:ADDR
value: int = driver.system.communicate.gpib.self.addr.get(gpibInstance = repcap.
↪GpibInstance.Default)
```

Sets the primary GPIB address of the analyzer.

param gpibInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Gpib')

return
adress_no: decimal Range: 0 to 30

set(*adress_no: int*, *gpibInstance=GpibInstance.Default*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB<inst>[:SELF]:ADDR
driver.system.communicate.gpib.self.addr.set(adress_no = 1, gpibInstance =␣
↪repcap.GpibInstance.Default)
```

Sets the primary GPIB address of the analyzer.

param adress_no
decimal Range: 0 to 30

param gpibInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Gpib')

6.27.2.1.1.2 Enable

SCPI Command :

```
SYSTem:COMMunicate:GPIB<inst>[:SELF]:ENABle
```

class EnableCls

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

get(*gpibInstance=GpibInstance.Default*) → bool

```
# SCPI: SYSTem:COMMunicate:GPIB<inst>[:SELF]:ENABle
value: bool = driver.system.communicate.gpib.self.enable.get(gpibInstance =␣
↪repcap.GpibInstance.Default)
```

Enables or disables the GPIB interface.

param gpibInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Gpib')

return
enable: No help available

set(*enable: bool*, *gpibInstance=GpibInstance.Default*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB<inst>[:SELF]:ENABle
driver.system.communicate.gpib.self.enable.set(enable = False, gpibInstance =␣
↪repcap.GpibInstance.Default)
```

Enables or disables the GPIB interface.

param enable
No help available

param gpibInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Gpib')

6.27.2.1.2 Vresource

SCPI Command :

```
SYSTem:COMMunicate:GPIB<inst>:VRESource
```

class VresourceCls

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

get(*gpibInstance=GpibInstance.Default*) → str

```
# SCPI: SYSTem:COMMunicate:GPIB<inst>:VRESource
value: str = driver.system.communicate.gpib.vresource.get(gpibInstance = repcap.
↪GpibInstance.Default)
```

Queries the VISA resource string of the GPIB interface.

param gpibInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Gpib')

return
visa_resource: No help available

6.27.2.2 Hislip<HislipInstance>

RepCap Settings

```
# Range: Inst1 .. Inst32
rc = driver.system.communicate.hislip.repcap_hislipInstance_get()
driver.system.communicate.hislip.repcap_hislipInstance_set(repcap.HislipInstance.Inst1)
```

class HislipCls

Hislip commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: HislipInstance, default value after init: HislipInstance.Inst1

Cloning the Group

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

Subgroups

6.27.2.2.1 Vresource

SCPI Command :

```
SYSTem:COMMunicate:HISLip<inst>:VRESource
```

class VresourceCls

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

get(*hislipInstance=HislipInstance.Default*) → str

```
# SCPI: SYSTem:COMMunicate:HISLip<inst>:VRESource
value: str = driver.system.communicate.hislip.vresource.get(hislipInstance =␣
↪repcap.HislipInstance.Default)
```

Queries the VISA resource string for the HiSLIP protocol.

param hislipInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Hislip')

return
visa_resource: No help available

6.27.2.3 Net

SCPI Commands :

```
SYSTem:COMMunicate:NET:ADAPter
SYSTem:COMMunicate:NET:GATeway
SYSTem:COMMunicate:NET:IPADdress
SYSTem:COMMunicate:NET:HOSTname
SYSTem:COMMunicate:NET:DHCP
```

class NetCls

Net commands group definition. 8 total commands, 2 Subgroups, 5 group commands

get_adapter() → str

```
# SCPI: SYSTem:COMMunicate:NET:ADAPter
value: str = driver.system.communicate.net.get_adapter()
```

Selects a LAN network adapter for configuration via other SYSTem:COMMunicate:NET... commands.

return
network_adapter: No help available

get_dhcp() → bool

```
# SCPI: SYSTem:COMMunicate:NET:DHCP
value: bool = driver.system.communicate.net.get_dhcp()
```

Enables or disables the dynamic host configuration protocol (DHCP) .

return
dhcp_enable: No help available

get_gateway() → List[str]

```
# SCPI: SYSTem:COMMunicate:NET:GATeway
value: List[str] = driver.system.communicate.net.get_gateway()
```

Manually defines IPv4 addresses of default gateways. A query returns the currently defined addresses, irrespective of whether they have been specified manually or via DHCP.

return
gateways: No help available

get_hostname() → str

```
# SCPI: SYSTem:COMMunicate:NET:HOSTname
value: str = driver.system.communicate.net.get_hostname()
```

Queries the host name (computer name) of the R&S CMW. The host name is part of the VISA address string for LAN-based connections.

return
hostname: string

get_ip_address() → List[str]

```
# SCPI: SYSTem:COMMunicate:NET:IPADdress
value: List[str] = driver.system.communicate.net.get_ip_address()
```

Manually assigns one or more IPv4 addresses to the network adapter. A query returns the currently assigned addresses, irrespective of whether they have been assigned manually or via DHCP.

return
ip_addresses: No help available

set_adapter(*network_adapter: str*) → None

```
# SCPI: SYSTem:COMMunicate:NET:ADAPter
driver.system.communicate.net.set_adapter(network_adapter = 'abc')
```

Selects a LAN network adapter for configuration via other SYSTem:COMMunicate:NET... commands.

param network_adapter
string

set_dhcp(*dhcp_enable: bool*) → None

```
# SCPI: SYSTem:COMMunicate:NET:DHCP
driver.system.communicate.net.set_dhcp(dhcp_enable = False)
```

Enables or disables the dynamic host configuration protocol (DHCP) .

param dhcp_enable
ON | OFF | 1 | 0 ON | 1: DHCP enabled, automatic TCP/IP address configuration. OFF | 0: DHCP disabled, manual address configuration.

set_gateway(*gateways: List[str]*) → None

```
# SCPI: SYSTem:COMMunicate:NET:GATeway
driver.system.communicate.net.set_gateway(gateways = ['abc1', 'abc2', 'abc3'])
```

Manually defines IPv4 addresses of default gateways. A query returns the currently defined addresses, irrespective of whether they have been specified manually or via DHCP.

param gateways
string Gateway IPv4 address consisting of four blocks separated by dots Several strings

separated by commas can be entered or several addresses separated by commas can be included in one string.

set_hostname(*hostname: str*) → None

```
# SCPI: SYSTem:COMMunicate:NET:HOSTname
driver.system.communicate.net.set_hostname(hostname = 'abc')
```

Queries the host name (computer name) of the R&S CMW. The host name is part of the VISA address string for LAN-based connections.

param hostname
string

set_ip_address(*ip_addresses: List[str]*) → None

```
# SCPI: SYSTem:COMMunicate:NET:IPADdress
driver.system.communicate.net.set_ip_address(ip_addresses = ['abc1', 'abc2',
↪'abc3'])
```

Manually assigns one or more IPv4 addresses to the network adapter. A query returns the currently assigned addresses, irrespective of whether they have been assigned manually or via DHCP.

param ip_addresses
string IPv4 address consisting of four blocks (octets) separated by dots Several strings separated by commas can be entered or several addresses separated by commas can be included in one string.

Cloning the Group

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

Subgroups

6.27.2.3.1 Dns

SCPI Commands :

```
SYSTem:COMMunicate:NET:DNS:ENABle
SYSTem:COMMunicate:NET:DNS
```

class DnsCls

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

get_enable() → bool

```
# SCPI: SYSTem:COMMunicate:NET:DNS:ENABle
value: bool = driver.system.communicate.net.dns.get_enable()
```

Enables or disables dynamic configuration of DNS server addresses.

return
dns_enable: No help available

get_value() → List[str]

```
# SCPI: SYSTem:COMMunicate:NET:DNS
value: List[str] = driver.system.communicate.net.dns.get_value()
```

Manually defines the DNS server IPv4 addresses to be used. A query returns the defined DNS addresses, irrespective of whether they have been specified manually or via DHCP.

return
ip_addresses: No help available

set_enable(*dns_enable: bool*) → None

```
# SCPI: SYSTem:COMMunicate:NET:DNS:ENABle
driver.system.communicate.net.dns.set_enable(dns_enable = False)
```

Enables or disables dynamic configuration of DNS server addresses.

param dns_enable
ON | OFF | 1 | 0 ON | 1: Enabled, automatic configuration OFF | 0: Disabled, manual configuration

set_value(*ip_addresses: List[str]*) → None

```
# SCPI: SYSTem:COMMunicate:NET:DNS
driver.system.communicate.net.dns.set_value(ip_addresses = ['abc1', 'abc2',
↪'abc3'])
```

Manually defines the DNS server IPv4 addresses to be used. A query returns the defined DNS addresses, irrespective of whether they have been specified manually or via DHCP.

param ip_addresses
string DNS server IPv4 addresses consisting of four blocks separated by dots. Several strings separated by commas can be entered or several addresses separated by commas can be included in one string.

6.27.2.3.2 Subnet

SCPI Command :

```
SYSTem:COMMunicate:NET:SUBNet:MASK
```

class SubnetCls

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

get_mask() → List[str]

```
# SCPI: SYSTem:COMMunicate:NET:SUBNet:MASK
value: List[str] = driver.system.communicate.net.subnet.get_mask()
```

Manually defines the subnet masks to be used for the network adapter IPv4 addresses. A query returns the currently used subnet masks, irrespective of whether they have been assigned manually or via DHCP.

return
subnet_masks: No help available

set_mask(*subnet_masks: List[str]*) → None

```
# SCPI: SYSTem:COMMunicate:NET:SUBNet:MASK
driver.system.communicate.net.subnet.set_mask(subnet_masks = ['abc1', 'abc2',
↪'abc3'])
```

Manually defines the subnet masks to be used for the network adapter IPv4 addresses. A query returns the currently used subnet masks, irrespective of whether they have been assigned manually or via DHCP.

param subnet_masks
string IPv4 subnet mask consisting of four blocks separated by dots Several strings separated by commas can be entered or several masks separated by commas can be included in one string.

6.27.2.4 Rsib<RsibInstance>

RepCap Settings

```
# Range: Inst1 .. Inst32
rc = driver.system.communicate.rsib.repcap_rsibInstance_get()
driver.system.communicate.rsib.repcap_rsibInstance_set(repcap.RsibInstance.Inst1)
```

class RsibCls

Rsib commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: RsibInstance, default value after init: RsibInstance.Inst1

Cloning the Group

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

Subgroups

6.27.2.4.1 Vresource

SCPI Command :

```
SYSTem:COMMunicate:RSIB<inst>:VRESource
```

class VresourceCls

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

get(*rsibInstance=RsibInstance.Default*) → str

```
# SCPI: SYSTem:COMMunicate:RSIB<inst>:VRESource
value: str = driver.system.communicate.rsib.vresource.get(rsibInstance = repcap.
↪RsibInstance.Default)
```

No command help available

param rsibInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Rsib')

return
visa_resource: No help available

6.27.2.5 Socket<SocketInstance>

RepCap Settings

```
# Range: Inst1 .. Inst32
rc = driver.system.communicate.socket.repcap_socketInstance_get()
driver.system.communicate.socket.repcap_socketInstance_set(repcap.SocketInstance.Inst1)
```

class SocketCls

Socket commands group definition. 3 total commands, 3 Subgroups, 0 group commands Repeated Capability: SocketInstance, default value after init: SocketInstance.Inst1

Cloning the Group

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

Subgroups

6.27.2.5.1 Mode

SCPI Command :

```
SYSTem:COMMunicate:SOCKet<inst>:MODE
```

class ModeCls

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

get(*socketInstance=SocketInstance.Default*) → SocketProtocol

```
# SCPI: SYSTem:COMMunicate:SOCKet<inst>:MODE
value: enums.SocketProtocol = driver.system.communicate.socket.mode.
↪get(socketInstance = repcap.SocketInstance.Default)
```

Sets the protocol operation mode for direct socket communication.

param socketInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Socket')

return
protocol_mode: No help available

set(*protocol_mode: SocketProtocol*, *socketInstance=SocketInstance.Default*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet<inst>:MODE
driver.system.communicate.socket.mode.set(protocol_mode = enums.SocketProtocol.
↪AGILent, socketInstance = repcap.SocketInstance.Default)
```

Sets the protocol operation mode for direct socket communication.

param protocol_mode
No help available

param socketInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Socket')

6.27.2.5.2 Port

SCPI Command :

```
SYSTem:COMMunicate:SOCKet<inst>:PORT
```

class PortCls

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

get(*socketInstance=SocketInstance.Default*) → int

```
# SCPI: SYSTem:COMMunicate:SOCKet<inst>:PORT
value: int = driver.system.communicate.socket.port.get(socketInstance = repcap.
↪SocketInstance.Default)
```

Sets the data port number for direct socket communication.

param socketInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Socket')

return
port_number: No help available

set(*port_number: int*, *socketInstance=SocketInstance.Default*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet<inst>:PORT
driver.system.communicate.socket.port.set(port_number = 1, socketInstance =␣
↪repcap.SocketInstance.Default)
```

Sets the data port number for direct socket communication.

param port_number
No help available

param socketInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Socket')

6.27.2.5.3 Vresource

SCPI Command :

```
SYSTem:COMMunicate:SOCKet<inst>:VRESource
```

class VresourceCls

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

get(*socketInstance=SocketInstance.Default*) → str

```
# SCPI: SYSTem:COMMunicate:SOCKet<inst>:VRESource
value: str = driver.system.communicate.socket.vresource.get(socketInstance =␣
↪repcap.SocketInstance.Default)
```

Queries the VISA resource string for direct socket communication.

param socketInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Socket')

return
visa_resource: No help available

6.27.2.6 Usb

SCPI Command :

```
SYSTem:COMMunicate:USB:VRESource
```

class UsbCls

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

get_vresource() → str

```
# SCPI: SYSTem:COMMunicate:USB:VRESource
value: str = driver.system.communicate.usb.get_vresource()
```

Queries the VISA resource string of the USB interface.

return
visa_resource: string

6.27.2.7 Vxi<VxiInstance>

RepCap Settings

```
# Range: Inst1 .. Inst32
rc = driver.system.communicate.vxi.repcap_vxiInstance_get()
driver.system.communicate.vxi.repcap_vxiInstance_set(repcap.VxiInstance.Inst1)
```

class VxiCls

Vxi commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: VxiInstance, default value after init: VxiInstance.Inst1

Cloning the Group

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

Subgroups

6.27.2.7.1 Gtr

SCPI Command :

```
SYSTem:COMMunicate:VXI<inst>:GTR
```

class GtrCls

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

get(*vxiInstance=VxiInstance.Default*) → bool

```
# SCPI: SYSTem:COMMunicate:VXI<inst>:GTR
value: bool = driver.system.communicate.vxi.gtr.get(vxiInstance = repcap.
↪VxiInstance.Default)
```

Enables or disables the VXI-11 interface.

param vxiInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Vxi')

return
bool_switchremote: No help available

set(*bool_switchremote: bool*, *vxiInstance=VxiInstance.Default*) → None

```
# SCPI: SYSTem:COMMunicate:VXI<inst>:GTR
driver.system.communicate.vxi.gtr.set(bool_switchremote = False, vxiInstance =␣
↪repcap.VxiInstance.Default)
```

Enables or disables the VXI-11 interface.

param bool_switchremote
No help available

param vxiInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Vxi')

6.27.2.7.2 Vresource

SCPI Command :

```
SYSTem:COMMunicate:VXI<inst>:VRESource
```

class VresourceCls

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

get(*vxiInstance=VxiInstance.Default*) → str

```
# SCPI: SYSTem:COMMunicate:VXI<inst>:VRESource
value: str = driver.system.communicate.vxi.vresource.get(vxiInstance = repcap.
↪VxiInstance.Default)
```

Queries the VISA resource string for the VXI-11 protocol.

param vxiInstance
optional repeated capability selector. Default value: Inst1 (settable in the interface 'Vxi')

return
visa_resource: No help available

6.27.3 Connector

class ConnectorCls

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

Cloning the Group

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

Subgroups

6.27.3.1 Translation

SCPI Command :

```
SYSTem:CONNector:TRANslation
```

class TranslationCls

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

class GetStruct

Response structure. Fields:

- Virtual_Connector: str: Returned virtual connector name
- Absolute_Connector: str: Returned physical connector name

get(*connector: str*) → GetStruct

```
# SCPI: SYSTem:CONNector:TRANslation
value: GetStruct = driver.system.connector.translation.get(connector = rawAbc)
```

Queries the relation between physical and virtual connector names. You can query this relation for either a physical or a virtual name. As a result, both physical and virtual name are returned. For background information and possible connector values, see 'Values for RF path selection'.

param connector
Physical or virtual connector name to be queried

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

6.27.4 Date

SCPI Command :

```
SYSTem:DATE
```

class DateCls

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

class DateStruct

Response structure. Fields:

- Year: int: No parameter help available
- Month: int: No parameter help available
- Day: int: No parameter help available

get() → DateStruct

```
# SCPI: SYSTem:DATE
value: DateStruct = driver.system.date.get()
```

No command help available

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

set(*year: int*, *month: int*, *day: int*) → None

```
# SCPI: SYSTem:DATE
driver.system.date.set(year = 1, month = 1, day = 1)
```

No command help available

param year
No help available

param month
No help available

param day
No help available

Cloning the Group

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

Subgroups

6.27.4.1 Local

SCPI Command :

```
SYSTem:DATE:LOCal
```

class LocalCls

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

class LocalStruct

Response structure. Fields:

- Year: int: No parameter help available
- Month: int: No parameter help available
- Day: int: No parameter help available

get() → LocalStruct

```
# SCPI: SYSTem:DATE:LOCal
value: LocalStruct = driver.system.date.local.get()
```

No command help available

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

set(*year: int*, *month: int*, *day: int*) → None

```
# SCPI: SYSTem:DATE:LOCal
driver.system.date.local.set(year = 1, month = 1, day = 1)
```

No command help available

param year
No help available

param month
No help available

param day
No help available

6.27.4.2 Utc

SCPI Command :

```
SYSTem:DATE:UTC
```

class UtcCls

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

class UtcStruct

Response structure. Fields:

- Year: int: No parameter help available
- Month: int: No parameter help available
- Day: int: No parameter help available

get() → UtcStruct

```
# SCPI: SYSTem:DATE:UTC
value: UtcStruct = driver.system.date.utc.get()
```

No command help available

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

set(*year: int*, *month: int*, *day: int*) → None

```
# SCPI: SYSTem:DATE:UTC
driver.system.date.utc.set(year = 1, month = 1, day = 1)
```

No command help available

param year
No help available

param month
No help available

param day
No help available

6.27.5 Device

SCPI Commands :

```
SYSTem:BASE:DEVice:SUBinst
SYSTem:BASE:DEVice:COUNt
SYSTem:BASE:DEVice:RESet
SYSTem:BASE:DEVice:MSCont
SYSTem:BASE:DEVice:MSCCount
SYSTem:DEVice:ID
```

class DeviceCls

Device commands group definition. 8 total commands, 2 Subgroups, 6 group commands

class SubinstStruct

Structure for reading output parameters. Fields:

- Cur_Sub_Inst: int: decimal Number of the addressed subinstrument, as indicated in a VISA resource string for VXI-11 Value n means instrument n+1. Example: 0 means instrument 1.
- Sub_Inst_Count: int: decimal Total number of subinstruments into which the instrument is split.

get_count() → int

```
# SCPI: SYSTem:BASE:DEVice:COUNt
value: int = driver.system.device.get_count()
```

Splits the instrument into subinstruments or assigns all hardware resources to a single subinstrument. Send this command to the subinstrument with the lowest number (device number 0 / assigned instrument 1 / subinstrument 1) . To assign/distribute the available hardware resources to the subinstruments, enter method RsCmwBase.System.Device.reset after you have changed the number of subinstruments.

return
count: integer Number of subinstruments The allowed values depend on your instrument configuration.

get_id() → str

```
# SCPI: SYSTem:DEVice:ID
value: str = driver.system.device.get_id()
```

Queries the device identification of the instrument. This ID is important for ordering licenses.

return
device_id: string

get_msc_count() → int

```
# SCPI: SYSTem:BASE:DEVice:MSCCount
value: int = driver.system.device.get_msc_count()
```

No command help available

return
max_sc_count: No help available

get_mscont() → int

```
# SCPI: SYSTem:BASE:DEVice:MSCont
value: int = driver.system.device.get_mscont()
```

Returns the maximum number of subinstruments into which the instrument can be split.

return
max_si_count: decimal Maximum number of subinstruments

get_subinst() → SubinstStruct

```
# SCPI: SYSTem:BASE:DEVice:SUBinst
value: SubinstStruct = driver.system.device.get_subinst()
```

Queries the number of the addressed subinstrument and the total number of subinstruments.

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

reset() → None

```
# SCPI: SYSTem:BASE:DEVice:RESet
driver.system.device.reset()
```

Assigns the available hardware resources to the subinstruments. Send this command to the subinstrument with the lowest number (device number 0 / assigned instrument 1 / subinstrument 1) . After changing the number of subinstruments via method RsCmwBase.System.Device.count, always send this command.

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

```
# SCPI: SYSTem:BASE:DEVice:RESet
driver.system.device.reset_with_opc()
```

Assigns the available hardware resources to the subinstruments. Send this command to the subinstrument with the lowest number (device number 0 / assigned instrument 1 / subinstrument 1) . After changing the number of subinstruments via method RsCmwBase.System.Device.count, always send this command.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCmwBase.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_count(*count: int*) → None

```
# SCPI: SYSTem:BASE:DEVice:COUNt
driver.system.device.set_count(count = 1)
```

Splits the instrument into subinstruments or assigns all hardware resources to a single subinstrument. Send this command to the subinstrument with the lowest number (device number 0 / assigned instrument 1 / subinstrument 1) . To assign/distribute the available hardware resources to the subinstruments, enter method RsCmwBase.System.Device.reset after you have changed the number of subinstruments.

param count
integer Number of subinstruments The allowed values depend on your instrument configuration.

Cloning the Group

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

Subgroups

6.27.5.1 License

SCPI Command :

```
SYSTem:BASE:DEVice:LICense
```

class LicenseCls

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

class LicenseStruct

Response structure. Fields:

- Sw_Option: List[str]: No parameter help available
- License_Count: List[int]: No parameter help available
- Instrument: List[int]: No parameter help available

get() → LicenseStruct

```
# SCPI: SYSTem:BASE:DEVice:LICense
value: LicenseStruct = driver.system.device.license.get()
```

No command help available

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

set(*sw_option: List[str] = None*, *license_count: List[int] = None*, *instrument: List[int] = None*) → None

```
# SCPI: SYSTem:BASE:DEVice:LICense
driver.system.device.license.set(sw_option = ['abc1', 'abc2', 'abc3'], license_
↪count = [1, 2, 3], instrument = [1, 2, 3])
```

No command help available

param sw_option
No help available

param license_count
No help available

param instrument
No help available

6.27.5.2 Setup

SCPI Command :

```
SYSTem:BASE:DEVice:SETup
```

class SetupCls

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

class SetupStruct

Response structure. Fields:

- Absolute_Item_Name: List[str]: No parameter help available
- Instrument: List[int]: No parameter help available

get() → SetupStruct

```
# SCPI: SYSTem:BASE:DEVice:SETup
value: SetupStruct = driver.system.device.setup.get()
```

No command help available

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

set(*absolute_item_name: List[str] = None*, *instrument: List[int] = None*) → None

```
# SCPI: SYSTem:BASE:DEVice:SETup
driver.system.device.setup.set(absolute_item_name = ['abc1', 'abc2', 'abc3'],␣
↪instrument = [1, 2, 3])
```

No command help available

param absolute_item_name
No help available

param instrument
No help available

6.27.6 DeviceFootprint

SCPI Command :

```
SYSTem:DFPRint
```

class DeviceFootprintCls

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

get() → bytes

```
# SCPI: SYSTem:DFPRint
value: bytes = driver.system.deviceFootprint.get()
```

Generates an XML file with footprint information about the instrument.

return
xml_device_footprint: block Block data element containing the XML file contents.

set(*path: str = None*) → None

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

Generates an XML file with footprint information about the instrument.

param path
No help available

6.27.7 Display

SCPI Commands :

```
SYSTem:BASE:DISPlay:MWINdow
SYSTem:BASE:DISPlay:COLorset
SYSTem:BASE:DISPlay:FONTset
SYSTem:BASE:DISPlay:ROLLkeymode
SYSTem:BASE:DISPlay:LANGuage
SYSTem:DISPlay:UPDate
```

class DisplayCls

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

get_color_set() → ColorSet

```
# SCPI: SYSTem:BASE:DISPlay:COLorset
value: enums.ColorSet = driver.system.display.get_color_set()
```

No command help available

return
color_set: No help available

get_font_set() → FontType

```
# SCPI: SYSTem:BASE:DISPlay:FONTset
value: enums.FontType = driver.system.display.get_font_set()
```

Selects the font size for the GUI labels.

return
fonset: No help available

get_language() → DisplayLanguage

```
# SCPI: SYSTem:BASE:DISPlay:LANGuage
value: enums.DisplayLanguage = driver.system.display.get_language()
```

No command help available

return
language: No help available

get_mwindow() → bool

```
# SCPI: SYSTem:BASE:DISPlay:MWINdow
value: bool = driver.system.display.get_mwindow()
```

Enables or disables the multiple-window mode of the graphical user interface.

return
on_off: No help available

get_rollkey_mode() → RollkeyMode

```
# SCPI: SYSTem:BASE:DISPlay:ROLLkeymode
value: enums.RollkeyMode = driver.system.display.get_rollkey_mode()
```

No command help available

return
rollkey_mode: No help available

get_update() → bool

```
# SCPI: SYSTem:DISPlay:UPDate
value: bool = driver.system.display.get_update()
```

Defines whether the display is updated or not while the instrument is in the remote state. If the display update is switched off, the normal GUI is replaced by a static image while the instrument is in the remote state. Switching off the display can speed up the measurement and is the recommended state. See also 'Using the display during remote control'

return
display_update: No help available

set_color_set(*color_set: ColorSet*) → None

```
# SCPI: SYSTem:BASE:DISPlay:COLorset
driver.system.display.set_color_set(color_set = enums.ColorSet.DEF)
```

No command help available

param color_set
No help available

set_font_set(*fonset: FontType*) → None

```
# SCPI: SYSTem:BASE:DISPlay:FONTset
driver.system.display.set_font_set(fonset = enums.FontType.DEF)
```

Selects the font size for the GUI labels.

param fonset
DEF | LRG DEF: Small fonts LRG: Large fonts

set_language(*language: DisplayLanguage*) → None

```
# SCPI: SYSTem:BASE:DISPlay:LANGuage
driver.system.display.set_language(language = enums.DisplayLanguage.AR)
```

No command help available

param language
No help available

set_mwindow(*on_off: bool*) → None

```
# SCPI: SYSTem:BASE:DISPlay:MWINdow
driver.system.display.set_mwindow(on_off = False)
```

Enables or disables the multiple-window mode of the graphical user interface.

param on_off
ON | OFF | 1 | 0 ON | 1: multiple-window mode OFF | 0: single-window mode

set_rollkey_mode(*rollkey_mode: RollkeyMode*) → None

```
# SCPI: SYSTem:BASE:DISPlay:ROLLkeymode
driver.system.display.set_rollkey_mode(rollkey_mode = enums.RollkeyMode.CURSors)
```

No command help available

param rollkey_mode
No help available

set_update(*display_update: bool*) → None

```
# SCPI: SYSTem:DISPlay:UPDate
driver.system.display.set_update(display_update = False)
```

Defines whether the display is updated or not while the instrument is in the remote state. If the display update is switched off, the normal GUI is replaced by a static image while the instrument is in the remote state. Switching off the display can speed up the measurement and is the recommended state. See also 'Using the display during remote control'

param display_update
ON | OFF | 1 | 0 ON | 1: Display is shown and updated during remote control. OFF | 0: Display shows static image during remote control.

Cloning the Group

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

Subgroups

6.27.7.1 Monitor

SCPI Command :

```
SYSTem:DISPlay:MONitor
```

class MonitorCls

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

set_value(*enable: bool*) → None

```
# SCPI: SYSTem:DISPlay:MONitor
driver.system.display.monitor.set_value(enable = False)
```

Turns the built-in display / the external monitor on or off.

param enable
ON | OFF

Cloning the Group

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

Subgroups

6.27.7.1.1 Off

SCPI Command :

```
SYSTem:DISPlay:MONitor:OFF
```

class OffCls

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

set() → None

```
# SCPI: SYSTem:DISPlay:MONitor:OFF
driver.system.display.monitor.off.set()
```

No command help available

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

```
# SCPI: SYSTem:DISPlay:MONitor:OFF
driver.system.display.monitor.off.set_with_opc()
```

No command help available

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

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

6.27.8 Error

SCPI Commands :

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

class ErrorCls

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

class AllStruct

Structure for reading output parameters. Fields:

- Error_Number: int: No parameter help available
- Error_Text: str: No parameter help available

get_all() → AllStruct

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

Queries and deletes all entries in the error queue.

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

get_count() → int

```
# SCPI: SYSTem:ERRor:COUNt
value: int = driver.system.error.get_count()
```

Queries the number of entries in the error queue.

return
error_count: integer

Cloning the Group

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

Subgroups

6.27.8.1 Code

SCPI Commands :

```
SYSTem:ERRor:CODE:ALL
SYSTem:ERRor:CODE[:NEXT]
```

class CodeCls

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

get_all() → int

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

Queries and deletes all entries in the error queue.

return
error_code: integer Comma-separated list of error numbers. The error descriptions are not returned. Positive error numbers are instrument-specific. Negative error numbers are reserved by the SCPI standard.

get_next() → int

```
# SCPI: SYSTem:ERRor:CODE[:NEXT]
value: int = driver.system.error.code.get_next()
```

Queries and deletes the oldest entry in the error queue.

return
error: integer Only the error number is returned, not the error description. Positive error numbers are instrument-specific. Negative error numbers are reserved by the SCPI standard.

6.27.9 Generator

class GeneratorCls

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

Cloning the Group

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

Subgroups

6.27.9.1 All

class AllCls

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

Cloning the Group

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

Subgroups

6.27.9.1.1 Off

SCPI Command :

```
SYSTem:GENerator:ALL:OFF
```

class OffCls

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

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

```
# SCPI: SYSTem:GENerator:ALL:OFF
driver.system.generator.all.off.set()
```

Switch off all signaling applications, generators or measurements.

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

6.27.10 Help

class HelpCls

Help commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.27.10.1 Headers

SCPI Command :

```
SYSTem:HELP:HEADers
```

class HeadersCls

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

get(*parser: str = None*) → bytes

```
# SCPI: SYSTem:HELP:HEADers
value: bytes = driver.system.help.headers.get(parser = 'abc')
```

No command help available

param parser
No help available

return
headers: No help available

6.27.10.2 Status

SCPI Commands :

```
SYSTem:HELP:STATus:BITS
SYSTem:HELP:STATus[:REGister]
```

class StatusCls

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

get_bits() → List[str]

```
# SCPI: SYSTem:HELP:STATus:BITS
value: List[str] = driver.system.help.status.get_bits()
```

Returns a list of paths for the bits of the STATus:OPERation registers at the lowest level of the hierarchy.

return
bits: string Comma-separated list of strings, one string per path The string for a path contains all registers from highest to lowest level separated by colons. Example: 'STATus:OPERation:TASK:A:GPRF:MEASurement:POWer:OFF'

get_register() → List[str]

```
# SCPI: SYSTem:HELP:STATus[:REGister]
value: List[str] = driver.system.help.status.get_register()
```

Returns a list of paths for the STATus:OPERation registers.

return
register: string Comma-separated list of strings, one string per path The string for a path contains all registers from highest level down to the individual register, separated by colons. For the GPRF power measurement, for example, the following paths are listed: 'STATus:OPERation', 'STATus:OPERation:TASK', 'STATus:OPERation:TASK:A', 'STATus:OPERation:TASK:A:GPRF', 'STATus:OPERation:TASK:A:GPRF:MEASurement', 'STATus:OPERation:TASK:A:GPRF:MEASurement:POWer'

6.27.10.3 Syntax

SCPI Commands :

```
SYSTem:HELP:SYNTax
SYSTem:HELP:SYNTax:ALL
```

class SyntaxCls

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

get(*header: str*) → bytes

```
# SCPI: SYSTem:HELP:SYNTax
value: bytes = driver.system.help.syntax.get(header = 'abc')
```

No command help available

param header
No help available

return
syntax: No help available

get_all() → bytes

```
# SCPI: SYSTem:HELP:SYNTax:ALL
value: bytes = driver.system.help.syntax.get_all()
```

No command help available

return
syntax: No help available

6.27.11 IpSet

class IpSetCls

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

Cloning the Group

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

Subgroups

6.27.11.1 SubMonitor

class SubMonitorCls

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

Cloning the Group

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

Subgroups

6.27.11.1.1 Refresh

SCPI Command :

```
SYSTem:BASE:IPSet:SMONitor:REFResh
```

class RefreshCls

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

set() → None

```
# SCPI: SYSTem:BASE:IPSet:SMONitor:REFResh
driver.system.ipSet.subMonitor.refresh.set()
```

Initiates an update of the information provided by the subnet monitor.

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

```
# SCPI: SYSTem:BASE:IPSet:SMONitor:REFResh
driver.system.ipSet.subMonitor.refresh.set_with_opc()
```

Initiates an update of the information provided by the subnet monitor.

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

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

6.27.12 Measurement

class MeasurementCls

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

Cloning the Group

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

Subgroups

6.27.12.1 All

class AllCls

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

Cloning the Group

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

Subgroups

6.27.12.1.1 Off

SCPI Command :

```
SYSTem:MEASurement:ALL:OFF
```

class OffCls

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

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

```
# SCPI: SYSTem:MEASurement:ALL:OFF
driver.system.measurement.all.off.set()
```

Switch off all signaling applications, generators or measurements.

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

6.27.13 Option

class OptionCls

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

Cloning the Group

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

Subgroups

6.27.13.1 Description

SCPI Command :

```
SYSTem:BASE:OPTion:DESCription
```

class DescriptionCls

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

get(*product_type: ProductType = None*, *validity: ValidityScope = None*, *scope: ValidityScopeB = None*, *instrument_no: float = None*) → str

```
# SCPI: SYSTem:BASE:OPTion:DESCription
value: str = driver.system.option.description.get(product_type = enums.
↪ProductType.ALL, validity = enums.ValidityScope.ALL, scope = enums.
↪ValidityScopeB.INSTrument, instrument_no = 1.0)
```

No command help available

param product_type
No help available

param validity
No help available

param scope
No help available

param instrument_no
No help available

return
option_list: No help available

6.27.13.2 ListPy

SCPI Command :

```
SYSTem:BASE:OPTion:LIST
```

class ListPyCls

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

get(*product_type: ProductType = None*, *validity: ValidityScope = None*, *scope: ValidityScopeB = None*, *instrument_no: float = None*) → str

```
# SCPI: SYSTem:BASE:OPTion:LIST
value: str = driver.system.option.listPy.get(product_type = enums.ProductType.
↪ALL, validity = enums.ValidityScope.ALL, scope = enums.ValidityScopeB.
↪INSTrument, instrument_no = 1.0)
```

Returns a list of installed software options (licenses) , hardware options, software packages and firmware applications. The list can be filtered using the described parameters. If filtering results in an empty list, a '0' is returned.

INTRO_CMD_HELP: The meaning of the filter <Validity> depends on the <OptionType> as follows:

- A software option is valid if there is an active license key for it. The value 'FUNCtional' is not relevant.
- A hardware option is functional if the corresponding hardware and all its components can be used (no defect detected) . The value 'VALid' is not relevant.
- A firmware application is functional if the required hardware, software and license keys are available and functional. The value 'VALid' is not relevant.
- For software packages, the filter has no effect.

param product_type
No help available

param validity
FUNCtional | VALid | ALL List only functional entries or only valid entries. By default or if ALL is selected, the list is not filtered according to the validity.

param scope
No help available

param instrument_no
No help available

return
option_list: No help available

6.27.13.3 Version

SCPI Command :

```
SYSTem:BASE:OPTion:VERSion
```

class VersionCls

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

get(*applicname: str = None*) → str

```
# SCPI: SYSTem:BASE:OPTion:VERSion
value: str = driver.system.option.version.get(applicname = 'abc')
```

Returns version information for installed software packages.

INTRO_CMD_HELP: You can either query a list of all installed packages and their versions or you can query the version of a single package specified via <Application>:

- <Application> specified: A string is returned, indicating the version of the <Application>. If the specified <Application> is unknown / not installed, '0' is returned.
- <Application> omitted: A string is returned, containing a list of all installed software packages and their version in the format '<PackageName1>,<Version1>;<PackageName2>,<Version2>;...'

param applicname
No help available

return
option_list: No help available

6.27.14 Password

SCPI Command :

```
SYSTem:BASE:PASSword:CDISable
```

class PasswordCls

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

set_cdisable(*user_mode: UserRole*) → None

```
# SCPI: SYSTem:BASE:PASSword:CDISable
driver.system.password.set_cdisable(user_mode = enums.UserRole.ADMin)
```

No command help available

param user_mode
No help available

Cloning the Group

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

Subgroups

6.27.14.1 Cenable

SCPI Commands :

```
SYSTem:BASE:PASSword[:CENable]:STATe
SYSTem:BASE:PASSword[:CENable]
```

class CenableCls

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

get_state() → UserRole

```
# SCPI: SYSTem:BASE:PASSword[:CENable]:STATe
value: enums.UserRole = driver.system.password.cenable.get_state()
```

No command help available

return
user_mode: No help available

set(*user_mode: UserRole*, *password: str*) → None

```
# SCPI: SYSTem:BASE:PASSword[:CENable]
driver.system.password.cenable.set(user_mode = enums.UserRole.ADMin, password =
↪'abc')
```

No command help available

param user_mode
No help available

param password
No help available

6.27.14.2 New

SCPI Command :

```
SYSTem:PASSword:NEW
```

class NewCls

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

set(*current_password: str*, *new_password: str*) → None

```
# SCPI: SYSTem:PASSword:NEW
driver.system.password.new.set(current_password = 'abc', new_password = 'abc')
```

No command help available

param current_password
No help available

param new_password
No help available

6.27.15 Record

class RecordCls

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

Cloning the Group

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

Subgroups

6.27.15.1 Macro

class MacroCls

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

Cloning the Group

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

Subgroups

6.27.15.1.1 File

SCPI Commands :

```
SYSTem:RECord:MACRo:FILE:STARt
SYSTem:RECord:MACRo:FILE:STOP
```

class FileCls

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

start(*macro_id: str*) → None

```
# SCPI: SYSTem:RECord:MACRo:FILE:STARt
driver.system.record.macro.file.start(macro_id = 'abc')
```

Starts recording of submitted commands into a macro file. If the file exists, it is overwritten. If the file does not exist, it is created.

param macro_id
string Path and filename of the destination file on the instrument

stop() → None

```
# SCPI: SYSTem:RECord:MACRo:FILE:STOP
driver.system.record.macro.file.stop()
```

Stops recording of commands into a macro file.

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

```
# SCPI: SYSTem:RECord:MACRo:FILE:STOP
driver.system.record.macro.file.stop_with_opc()
```

Stops recording of commands into a macro file.

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

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

6.27.16 Reference

class ReferenceCls

Reference commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.27.16.1 Dc

class DcCls

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

Cloning the Group

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

Subgroups

6.27.16.1.1 Offset

SCPI Command :

```
SYSTem:BASE:REFerence:DC:OFFSet:ENABle
```

class OffsetCls

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

get_enable() → bool

```
# SCPI: SYSTem:BASE:REFerence:DC:OFFSet:ENABle
value: bool = driver.system.reference.dc.offset.get_enable()
```

No command help available

return
dc_offset_enable: No help available

set_enable(*dc_offset_enable: bool*) → None

```
# SCPI: SYSTem:BASE:REFerence:DC:OFFSet:ENABle
driver.system.reference.dc.offset.set_enable(dc_offset_enable = False)
```

No command help available

param dc_offset_enable
No help available

6.27.16.2 Frequency<Frequency>

RepCap Settings

```
# Range: Freq1 .. Freq4
rc = driver.system.reference.frequency.repcap_frequency_get()
driver.system.reference.frequency.repcap_frequency_set(repcap.Frequency.Freq1)
```

SCPI Commands :

```
SYSTem:BASE:REFerence:FREQuency:SOURce
SYSTem:BASE:REFerence:FREQuency
```

class FrequencyCls

Frequency commands group definition. 3 total commands, 1 Subgroups, 2 group commands Repeated Capability: Frequency, default value after init: Frequency.Freq1

get_source() → SourceIntExt

```
# SCPI: SYSTem:BASE:REFerence:FREQuency:SOURce
value: enums.SourceIntExt = driver.system.reference.frequency.get_source()
```

Selects the reference frequency source to be used.

return
source: No help available

get_value() → float

```
# SCPI: SYSTem:BASE:REFerence:FREQuency
value: float = driver.system.reference.frequency.get_value()
```

Sets the R&S CMW external reference frequency.

return
ref_frequency: No help available

set_source(*source: SourceIntExt*) → None

```
# SCPI: SYSTem:BASE:REFerence:FREQuency:SOURce
driver.system.reference.frequency.set_source(source = enums.SourceIntExt.
↪EINTernal)
```

Selects the reference frequency source to be used.

param source
INTernal | EXTernal INTernal: Internal reference frequency EXTernal: External reference frequency

set_value(*ref_frequency: float*) → None

```
# SCPI: SYSTem:BASE:REFerence:FREQuency
driver.system.reference.frequency.set_value(ref_frequency = 1.0)
```

Sets the R&S CMW external reference frequency.

param ref_frequency
numeric Range: 1 MHz to 80 MHz, Unit: Hz

Cloning the Group

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

Subgroups

6.27.16.2.1 Advanced

class AdvancedCls

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

Cloning the Group

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

Subgroups

6.27.16.2.1.1 Source

SCPI Command :

```
SYSTem:BASE:REFerence:FREQuency<n>:ADVanced:SOURce
```

class SourceCls

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

get(*frequency=Frequency.Default*) → SourceIntExt

```
# SCPI: SYSTem:BASE:REFerence:FREQuency<n>:ADVanced:SOURce
value: enums.SourceIntExt = driver.system.reference.frequency.advanced.source.
↪get(frequency = repcap.Frequency.Default)
```

No command help available

param frequency
optional repeated capability selector. Default value: Freq1 (settable in the interface 'Frequency')

return
source: No help available

set(*source: SourceIntExt*, *frequency=Frequency.Default*) → None

```
# SCPI: SYSTem:BASE:REFerence:FREQuency<n>:ADVanced:SOURce
driver.system.reference.frequency.advanced.source.set(source = enums.
↪SourceIntExt.EINTernal, frequency = repcap.Frequency.Default)
```

No command help available

param source
No help available

param frequency
optional repeated capability selector. Default value: Freq1 (settable in the interface 'Frequency')

6.27.16.3 Phase

SCPI Command :

```
SYSTem:BASE:REFerence:PHASe:OFFSet
```

class PhaseCls

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

get_offset() → float

```
# SCPI: SYSTem:BASE:REFerence:PHASe:OFFSet
value: float = driver.system.reference.phase.get_offset()
```

No command help available

return
phase_offset: No help available

set_offset(*phase_offset: float*) → None

```
# SCPI: SYSTem:BASE:REFerence:PHASe:OFFSet
driver.system.reference.phase.set_offset(phase_offset = 1.0)
```

No command help available

param phase_offset
No help available

6.27.17 Routing

class RoutingCls

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

Cloning the Group

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

Subgroups

6.27.17.1 Possible

SCPI Command :

```
SYSTem:ROUTing:POSSible
```

class PossibleCls

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

get(*item: str = None*) → List[str]

```
# SCPI: SYSTem:ROUTing:POSSible
value: List[str] = driver.system.routing.possible.get(item = 'abc')
```

No command help available

param item
No help available

return
routing: No help available

6.27.18 Signaling

class SignalingCls

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

Cloning the Group

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

Subgroups

6.27.18.1 All

class AllCls

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

Cloning the Group

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

Subgroups

6.27.18.1.1 Off

SCPI Command :

```
SYSTem:SIGNaling:ALL:OFF
```

class OffCls

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

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

```
# SCPI: SYSTem:SIGNaling:ALL:OFF
driver.system.signaling.all.off.set()
```

Switch off all signaling applications, generators or measurements.

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

6.27.19 SingleCmw

class SingleCmwCls

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

Cloning the Group

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

Subgroups

6.27.19.1 Device

SCPI Command :

```
SYSTem:CMWS:DEVice:ID
```

class DeviceCls

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

get_id() → str

```
# SCPI: SYSTem:CMWS:DEVice:ID
value: str = driver.system.singleCmw.device.get_id()
```

No command help available

return
idn: No help available

6.27.20 Ssync

SCPI Commands :

```
SYSTem:BASE:SSYNc:MODE
SYSTem:BASE:SSYNc:OFFSet
```

class SsyncCls

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

get_mode() → CmwMode

```
# SCPI: SYSTem:BASE:SSYNc:MODE
value: enums.CmwMode = driver.system.ssync.get_mode()
```

Specifies the role of the instrument in a multi-CMW setup, concerning the system time synchronization signal.

return
mode: LIST | GEN | STAN LISTener: The instrument receives a time synchronization signal at 'SYS SYNC IN'. GENerator: The instrument provides a system synchronization signal at the rear panel. STANdalone: The instrument uses its internal synchronization signal.

get_offset() → int

```
# SCPI: SYSTem:BASE:SSYNc:OFFSet
value: int = driver.system.ssync.get_offset()
```

No command help available

return
offset: No help available

set_mode(*mode: CmwMode*) → None

```
# SCPI: SYSTem:BASE:SSYNc:MODE
driver.system.ssync.set_mode(mode = enums.CmwMode.GENerator)
```

Specifies the role of the instrument in a multi-CMW setup, concerning the system time synchronization signal.

param mode
LIST | GEN | STAN LISTener: The instrument receives a time synchronization signal at 'SYS SYNC IN'. GENerator: The instrument provides a system synchronization signal at the rear panel. STANdalone: The instrument uses its internal synchronization signal.

set_offset(*offset: int*) → None

```
# SCPI: SYSTem:BASE:SSYNc:OFFSet
driver.system.ssync.set_offset(offset = 1)
```

No command help available

param offset
No help available

6.27.21 Startup

class StartupCls

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

Cloning the Group

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

Subgroups

6.27.21.1 Prepare

SCPI Command :

```
SYSTem:STARtup:PREPare:FDEFault
```

class PrepareCls

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

get_fdefault() → bool

```
# SCPI: SYSTem:STARtup:PREPare:FDEFault
value: bool = driver.system.startup.prepare.get_fdefault()
```

Enables startup with factory default state.

return
on_off: No help available

set_fdefault(*on_off: bool*) → None

```
# SCPI: SYSTem:STARtup:PREPare:FDEFault
driver.system.startup.prepare.set_fdefault(on_off = False)
```

Enables startup with factory default state.

param on_off
ON | OFF | 1 | 0 Behavior during the startup: OFF | 0: Restore previous settings. ON | 1: Set instrument to its factory default state.

6.27.22 StIcon

SCPI Commands :

```
SYSTem:BASE:STICon:ENABle
SYSTem:BASE:STICon:OPEN
SYSTem:BASE:STICon:CLOSe
```

class StIconCls

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

close() → None

```
# SCPI: SYSTem:BASE:STICon:CLOSe
driver.system.stIcon.close()
```

Hides all windows and taskbar entries of the CMW application. Prerequisite: A CMW software icon has been added to the system tray (ENABle command) .

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

```
# SCPI: SYSTem:BASE:STICon:CLOSe
driver.system.stIcon.close_with_opc()
```

Hides all windows and taskbar entries of the CMW application. Prerequisite: A CMW software icon has been added to the system tray (ENABle command) .

Same as close, but waits for the operation to complete before continuing further. Use the RsCmwBase.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_enable() → bool

```
# SCPI: SYSTem:BASE:STICon:ENABle
value: bool = driver.system.stIcon.get_enable()
```

Selects whether an icon for the CMW software is added to the system tray of the operating system.

return
on_off: ON | OFF | 1 | 0 ON | 1: icon in system tray OFF | 0: no icon in system tray

open() → None

```
# SCPI: SYSTem:BASE:STICon:OPEN
driver.system.stIcon.open()
```

Restores the windows and taskbar entries of the CMW application after they have been hidden by the CLOSe command. Prerequisite: A CMW software icon has been added to the system tray (ENABle command) .

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

```
# SCPI: SYSTem:BASE:STICon:OPEN
driver.system.stIcon.open_with_opc()
```

Restores the windows and taskbar entries of the CMW application after they have been hidden by the CLOSe command. Prerequisite: A CMW software icon has been added to the system tray (ENABle command) .

Same as open, but waits for the operation to complete before continuing further. Use the RsCmwBase.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_enable(*on_off: bool*) → None

```
# SCPI: SYSTem:BASE:STICon:ENABle
driver.system.stIcon.set_enable(on_off = False)
```

Selects whether an icon for the CMW software is added to the system tray of the operating system.

param on_off
ON | OFF | 1 | 0 ON | 1: icon in system tray OFF | 0: no icon in system tray

6.27.23 Time

SCPI Command :

```
SYSTem:TIME
```

class TimeCls

Time commands group definition. 10 total commands, 4 Subgroups, 1 group commands

class TimeStruct

Response structure. Fields:

- Hour: int: No parameter help available
- Min_Py: int: No parameter help available
- Sec: int: No parameter help available

get() → TimeStruct

```
# SCPI: SYSTem:TIME
value: TimeStruct = driver.system.time.get()
```

No command help available

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

set(*hour: int*, *min_py: int*, *sec: int*) → None

```
# SCPI: SYSTem:TIME
driver.system.time.set(hour = 1, min_py = 1, sec = 1)
```

No command help available

param hour
No help available

param min_py
No help available

param sec
No help available

Cloning the Group

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

Subgroups

6.27.23.1 DaylightSavingTime

SCPI Command :

```
SYSTem:TIME:DSTime:MODE
```

class DaylightSavingTimeCls

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

get_mode() → bool

```
# SCPI: SYSTem:TIME:DSTime:MODE
value: bool = driver.system.time.daylightSavingTime.get_mode()
```

No command help available

return
dst: No help available

set_mode(*dst: bool*) → None

```
# SCPI: SYSTem:TIME:DSTime:MODE
driver.system.time.daylightSavingTime.set_mode(dst = False)
```

No command help available

param dst
No help available

Cloning the Group

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

Subgroups

6.27.23.1.1 Rule

SCPI Commands :

```
SYSTem:TIME:DSTime:RULE:CATalog
SYSTem:TIME:DSTime:RULE
```

class RuleCls

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

get_catalog() → str

```
# SCPI: SYSTem:TIME:DSTime:RULE:CATalog
value: str = driver.system.time.daylightSavingTime.rule.get_catalog()
```

No command help available

return
cat: No help available

get_value() → str

```
# SCPI: SYSTem:TIME:DSTime:RULE
value: str = driver.system.time.daylightSavingTime.rule.get_value()
```

No command help available

return
rule: No help available

set_value(*rule: str*) → None

```
# SCPI: SYSTem:TIME:DSTime:RULE
driver.system.time.daylightSavingTime.rule.set_value(rule = 'abc')
```

No command help available

param rule
No help available

6.27.23.2 HrTimer

SCPI Command :

```
SYSTem:TIME:HRTimer:RELative
```

class HrTimerCls

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

set_relative(*duration: int*) → None

```
# SCPI: SYSTem:TIME:HRTimer:RELative
driver.system.time.hrTimer.set_relative(duration = 1)
```

This command starts a timer. After the specified timeout, an OPC is generated. When the timer expires, 'Operation Complete' is indicated. This event can be evaluated by polling, via a *OPC? or via *WAI.

param duration
integer Range: 0 ms to 4294967295 ms, Unit: ms

Cloning the Group

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

Subgroups

6.27.23.2.1 Absolute

SCPI Commands :

```
SYSTem:TIME:HRTimer:ABSolute:CLEar
SYSTem:TIME:HRTimer:ABSolute
```

class AbsoluteCls

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

clear() → None

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:CLEar
driver.system.time.hrTimer.absolute.clear()
```

No command help available

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

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:CLEar
driver.system.time.hrTimer.absolute.clear_with_opc()
```

No command help available

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

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

set_value(*duration: float*) → None

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute
driver.system.time.hrTimer.absolute.set_value(duration = 1.0)
```

This command starts a timer. The timeout is specified relative to an already set timestamp, see method RsCmwBase.System. Time.HrTimer.Absolute.Set.set. When the timer expires, 'Operation Complete' is indicated. This event can be evaluated by polling, via a *OPC? or via *WAI.

param duration
integer Range: 0 ms to 4294967295 ms, Unit: ms

Cloning the Group

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

Subgroups

6.27.23.2.1.1 Set

SCPI Command :

```
SYSTem:TIME:HRTimer:ABSolute:SET
```

class SetCls

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

class GetStruct

Response structure. 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
- Min_Py: int: No parameter help available
- Sec: float: No parameter help available
- Msec: int: No parameter help available

get() → GetStruct

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:SET
value: GetStruct = driver.system.time.hrTimer.absolute.set.get()
```

This command sets a timestamp with the current system time. A timer can be started with a timeout relative to this timestamp, see method RsCmwBase.System.Time.HrTimer.Absolute.value. An existing timestamp is overwritten.

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

set() → None

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:SET
driver.system.time.hrTimer.absolute.set.set()
```

This command sets a timestamp with the current system time. A timer can be started with a timeout relative to this timestamp, see method RsCmwBase.System.Time.HrTimer.Absolute.value. An existing timestamp is overwritten.

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

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:SET
driver.system.time.hrTimer.absolute.set.set_with_opc()
```

This command sets a timestamp with the current system time. A timer can be started with a timeout relative to this timestamp, see method RsCmwBase.System.Time.HrTimer.Absolute.value. An existing timestamp is overwritten.

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

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

6.27.23.3 Local

SCPI Command :

```
SYSTem:TIME:LOCal
```

class LocalCls

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

class LocalStruct

Response structure. Fields:

- Hour: int: No parameter help available
- Minute: int: No parameter help available
- Second: int: No parameter help available

get() → LocalStruct

```
# SCPI: SYSTem:TIME:LOCal
value: LocalStruct = driver.system.time.local.get()
```

No command help available

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

set(*hour: int*, *minute: int*, *second: int*) → None

```
# SCPI: SYSTem:TIME:LOCal
driver.system.time.local.set(hour = 1, minute = 1, second = 1)
```

No command help available

param hour
No help available

param minute
No help available

param second
No help available

6.27.23.4 Utc

SCPI Command :

```
SYSTem:TIME:UTC
```

class UtcCls

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

class UtcStruct

Response structure. Fields:

- Hour: int: No parameter help available
- Minute: int: No parameter help available
- Second: int: No parameter help available

get() → UtcStruct

```
# SCPI: SYSTem:TIME:UTC
value: UtcStruct = driver.system.time.utc.get()
```

No command help available

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

set(*hour: int*, *minute: int*, *second: int*) → None

```
# SCPI: SYSTem:TIME:UTC
driver.system.time.utc.set(hour = 1, minute = 1, second = 1)
```

No command help available

param hour
No help available

param minute
No help available

param second
No help available

6.27.24 Tzone

SCPI Command :

```
SYSTem:TZONe
```

class TzoneCls

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

class TzoneStruct

Response structure. Fields:

- Hour: int: No parameter help available

- Minute: int: No parameter help available

get() → TzoneStruct

```
# SCPI: SYSTem:TZONe
value: TzoneStruct = driver.system.tzone.get()
```

No command help available

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

set(*hour: int*, *minute: int*) → None

```
# SCPI: SYSTem:TZONe
driver.system.tzone.set(hour = 1, minute = 1)
```

No command help available

param hour
No help available

param minute
No help available

6.27.25 Update

SCPI Command :

```
SYSTem:UPDate:DGRoup
```

class UpdateCls

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

get_dgroup() → str

```
# SCPI: SYSTem:UPDate:DGRoup
value: str = driver.system.update.get_dgroup()
```

Sets the 'Device Group' that the instrument belongs to. For remote installation, this setting must match the corresponding setting in the R&S Software Distributor options.

return
devicegroup: string

set_dgroup(*devicegroup: str*) → None

```
# SCPI: SYSTem:UPDate:DGRoup
driver.system.update.set_dgroup(devicegroup = 'abc')
```

Sets the 'Device Group' that the instrument belongs to. For remote installation, this setting must match the corresponding setting in the R&S Software Distributor options.

param devicegroup
string

6.28 Trace

class TraceCls

Trace commands group definition. 13 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.28.1 Remote

class RemoteCls

Remote commands group definition. 13 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.28.1.1 Mode

class ModeCls

Mode commands group definition. 13 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.28.1.1.1 Display

SCPI Commands :

```
TRACe:REMote:MODE:DISPlay:CLEar
TRACe:REMote:MODE:DISPlay:ENABle
```

class DisplayCls

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

clear() → None

```
# SCPI: TRACe:REMote:MODE:DISPlay:CLEar
driver.trace.remote.mode.display.clear()
```

Clears the display of the SCPI remote trace in analysis mode.

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

```
# SCPI: TRACe:REMote:MODE:DISPlay:CLEar
driver.trace.remote.mode.display.clear_with_opc()
```

Clears the display of the SCPI remote trace in analysis mode.

Same as clear, but waits for the operation to complete before continuing further. Use the RsCmwBase.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_enable() → RemoteTraceEnable

```
# SCPI: TRACe:REMote:MODE:DISPlay:ENABle
value: enums.RemoteTraceEnable = driver.trace.remote.mode.display.get_enable()
```

Enables or disables the display of the SCPI remote trace. Two modes are available when the display is enabled: a live mode and an analysis mode.

return
benable: No help available

set_enable(*benable: RemoteTraceEnable*) → None

```
# SCPI: TRACe:REMote:MODE:DISPlay:ENABle
driver.trace.remote.mode.display.set_enable(benable = enums.RemoteTraceEnable.
↪ANALysis)
```

Enables or disables the display of the SCPI remote trace. Two modes are available when the display is enabled: a live mode and an analysis mode.

param benable
ANALysis | LIVE | OFF ANALysis: Stop tracing to analyze already traced messages. LIVE: Trace messages and display them. OFF: Disable the report display. Default value: OFF

6.28.1.1.2 File<FileNr>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.trace.remote.mode.file.repcap_fileNr_get()
driver.trace.remote.mode.file.repcap_fileNr_set(repcap.FileNr.Nr1)
```

class FileCls

File commands group definition. 11 total commands, 11 Subgroups, 0 group commands Repeated Capability: FileNr, default value after init: FileNr.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.remote.mode.file.clone()
```

Subgroups

6.28.1.1.2.1 Dexecution

class DexecutionCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.remote.mode.file.dexecution.clone()
```

Subgroups

6.28.1.1.2.2 Duration

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:DEXecution:DURation
```

class DurationCls

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

get(*fileNr=FileNr.Default*) → bool

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:DEXecution:DURation
value: bool = driver.trace.remote.mode.file.dexecution.duration.get(fileNr =␣
↪repcap.FileNr.Default)
```

No command help available

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

return
benabled: No help available

set(*benabled: bool*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:DEXecution:DURation
driver.trace.remote.mode.file.dexecution.duration.set(benabled = False, fileNr␣
↪= repcap.FileNr.Default)
```

No command help available

param benabled
No help available

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

6.28.1.1.2.3 Enable

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:ENABle
```

class EnableCls

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

get(*fileNr=FileNr.Default*) → bool

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:ENABle
value: bool = driver.trace.remote.mode.file.enable.get(fileNr = repcap.FileNr.
↪Default)
```

Enable or disable tracing of the remote interface to a file for the specified subinstrument.

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

return
benable: No help available

set(*benable: bool*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:ENABle
driver.trace.remote.mode.file.enable.set(benable = False, fileNr = repcap.
↪FileNr.Default)
```

Enable or disable tracing of the remote interface to a file for the specified subinstrument.

param benable
No help available

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

6.28.1.1.2.4 FilterPy

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:FILTer
```

class FilterPyCls

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

class FilterPyStruct

Structure for setting input parameters. Fields:

- Binput: bool: No parameter help available
- Boutput: bool: No parameter help available
- Berror: bool: No parameter help available
- Btrigger: bool: No parameter help available
- Bdevice_Clear: bool: No parameter help available
- Bstatus_Register: bool: No parameter help available
- Bconnection: bool: No parameter help available
- Bremote_Local_Events: bool: No parameter help available

get(*fileNr=FileNr.Default*) → FilterPyStruct

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FILTer
value: FilterPyStruct = driver.trace.remote.mode.file.filterPy.get(fileNr =␣
↪repcap.FileNr.Default)
```

Specifies a filter for tracing of the specified subinstrument. The filter defines which message types and events are traced into a file. The default setting is ON,ON,ON,OFF,OFF,OFF,OFF,OFF.

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

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

set(*structure: FilterPyStruct*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FILTer
structure = driver.trace.remote.mode.file.filterPy.FilterPyStruct()
structure.Binput: bool = False
structure.Boutput: bool = False
structure.Berror: bool = False
structure.Btrigger: bool = False
structure.Bdevice_Clear: bool = False
structure.Bstatus_Register: bool = False
```

(continues on next page)

(continued from previous page)

```
structure.Bconnection: bool = False
structure.Bremote_Local_Events: bool = False
driver.trace.remote.mode.file.filterPy.set(structure, fileNr = repcap.FileNr.
↪Default)
```

Specifies a filter for tracing of the specified subinstrument. The filter defines which message types and events are traced into a file. The default setting is ON,ON,ON,OFF,OFF,OFF,OFF,OFF.

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

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

6.28.1.1.2.5 FormatPy

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:FORMat
```

class FormatPyCls

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

get(*fileNr=FileNr.Default*) → RemoteTraceFileFormat

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FORMat
value: enums.RemoteTraceFileFormat = driver.trace.remote.mode.file.formatPy.
↪get(fileNr = repcap.FileNr.Default)
```

Specifies the format of the target file for tracing of the remote interface for the specified subinstrument. The trace can be stored as ASCII file or as XML file.

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

return
eformat: No help available

set(*eformat: RemoteTraceFileFormat*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FORMat
driver.trace.remote.mode.file.formatPy.set(eformat = enums.
↪RemoteTraceFileFormat.ASCii, fileNr = repcap.FileNr.Default)
```

Specifies the format of the target file for tracing of the remote interface for the specified subinstrument. The trace can be stored as ASCII file or as XML file.

param eformat
No help available

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

6.28.1.1.2.6 Functions

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:FUNCtions
```

class FunctionsCls

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

get(*fileNr=FileNr.Default*) → bool

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FUNCtions
value: bool = driver.trace.remote.mode.file.functions.get(fileNr = repcap.
↪FileNr.Default)
```

No command help available

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

return
benable: No help available

set(*benable: bool*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FUNCtions
driver.trace.remote.mode.file.functions.set(benable = False, fileNr = repcap.
↪FileNr.Default)
```

No command help available

param benable
No help available

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

6.28.1.1.2.7 Name

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:NAME
```

class NameCls

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

get(*fileNr=FileNr.Default*) → str

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:NAME
value: str = driver.trace.remote.mode.file.name.get(fileNr = repcap.FileNr.
↪Default)
```

Specify path and name of the target file for tracing of the remote interface. For different subinstruments, specify different files. If you specify a new target file while tracing, the old target file is closed, the new file is created and tracing is continued with the new file.

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

return
bs_file_path: No help available

set(*bs_file_path: str*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:NAME
driver.trace.remote.mode.file.name.set(bs_file_path = 'abc', fileNr = repcap.
↪FileNr.Default)
```

Specify path and name of the target file for tracing of the remote interface. For different subinstruments, specify different files. If you specify a new target file while tracing, the old target file is closed, the new file is created and tracing is continued with the new file.

param bs_file_path
No help available

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

6.28.1.1.2.8 Parser

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:PARSer
```

class ParserCls

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

get(*fileNr=FileNr.Default*) → bool

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:PARSer
value: bool = driver.trace.remote.mode.file.parser.get(fileNr = repcap.FileNr.
↪Default)
```

No command help available

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

return
benable: No help available

set(*benable: bool*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:PARSer
driver.trace.remote.mode.file.parser.set(benable = False, fileNr = repcap.
↪FileNr.Default)
```

No command help available

param benable
No help available

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

6.28.1.1.2.9 Rpc

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:RPC
```

class RpcCls

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

get(*fileNr=FileNr.Default*) → bool

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:RPC
value: bool = driver.trace.remote.mode.file.rpc.get(fileNr = repcap.FileNr.
↪Default)
```

No command help available

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

return
benable: No help available

set(*benable: bool*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:RPC
driver.trace.remote.mode.file.rpc.set(benable = False, fileNr = repcap.FileNr.
↪Default)
```

No command help available

param benable
No help available

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

6.28.1.1.2.10 Size

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:SIZE
```

class SizeCls

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

get(*fileNr=FileNr.Default*) → int

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:SIZE
value: int = driver.trace.remote.mode.file.size.get(fileNr = repcap.FileNr.
↪Default)
```

Specifies the maximum size of the trace file in bytes.

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

return
ifile_size: No help available

set(*ifile_size: int*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:SIZE
driver.trace.remote.mode.file.size.set(ifile_size = 1, fileNr = repcap.FileNr.
↪Default)
```

Specifies the maximum size of the trace file in bytes.

param ifile_size
No help available

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

6.28.1.1.2.11 StartMode

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:STARtmode
```

class StartModeCls

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

get(*fileNr=FileNr.Default*) → RemoteTraceStartMode

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STARtmode
value: enums.RemoteTraceStartMode = driver.trace.remote.mode.file.startMode.
↪get(fileNr = repcap.FileNr.Default)
```

Specifies whether tracing is started automatically or manually.

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

return
estart_mode: AUTO | EXPLicit AUTO: Start tracing automatically when the instrument is started. EXPLicit: Start tracing via the command method RsCmwBase.Trace.Remote.Mode.File.Enable.set. Default value: EXPLicit

set(*estart_mode: RemoteTraceStartMode*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STARtmode
driver.trace.remote.mode.file.startMode.set(estart_mode = enums.
↪RemoteTraceStartMode.AUTO, fileNr = repcap.FileNr.Default)
```

Specifies whether tracing is started automatically or manually.

param estart_mode
AUTO | EXPLicit AUTO: Start tracing automatically when the instrument is started. EXPLicit: Start tracing via the command method RsCmwBase.Trace.Remote.Mode.File.Enable.set. Default value: EXPLicit

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

6.28.1.1.2.12 StopMode

SCPI Command :

```
TRACe:REMote:MODE:FILE<instrument>:STOPmode
```

class StopModeCls

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

get(*fileNr=FileNr.Default*) → RemoteTraceStopMode

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STOPmode
value: enums.RemoteTraceStopMode = driver.trace.remote.mode.file.stopMode.
↪get(fileNr = repcap.FileNr.Default)
```

Specifies how / when tracing is stopped and the trace file is closed.

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

return
estop_mode: AUTO | EXPLicit | ERRor | BUFFerfull AUTO: Stop tracing automatically when the instrument is shut down. EXPLicit: Stop tracing via the command method RsCmwBase.Trace.Remote.Mode.File.Enable.set. ERRor: Stop tracing when a SCPI error occurs. BUFFerfull: Stop tracing when the maximum file size is reached. Default value: EXPLicit

set(*estop_mode: RemoteTraceStopMode*, *fileNr=FileNr.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STOPmode
driver.trace.remote.mode.file.stopMode.set(estop_mode = enums.
↪RemoteTraceStopMode.AUTO, fileNr = repcap.FileNr.Default)
```

Specifies how / when tracing is stopped and the trace file is closed.

param estop_mode
AUTO | EXPLicit | ERRor | BUFFerfull AUTO: Stop tracing automatically when the instrument is shut down. EXPLicit: Stop tracing via the command method RsCmwBase.Trace.Remote.Mode.File.Enable.set. ERRor: Stop tracing when a SCPI error occurs. BUFFerfull: Stop tracing when the maximum file size is reached. Default value: EXPLicit

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

6.29 Trigger

class TriggerCls

Trigger commands group definition. 11 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.29.1 Eout<Eout>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.trigger.eout.repcap_eout_get()
driver.trigger.eout.repcap_eout_set(repcap.Eout.Nr1)
```

class EoutCls

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

Cloning the Group

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

Subgroups

6.29.1.1 Catalog

class CatalogCls

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

Cloning the Group

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

Subgroups

6.29.1.1.1 Source

SCPI Command :

```
TRIGger:BASE:EOUT<n>:CATalog:SOURce
```

class SourceCls

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

get(*eout=Eout.Default*) → List[str]

```
# SCPI: TRIGger:BASE:EOUT<n>:CATalog:SOURce
value: List[str] = driver.trigger.eout.catalog.source.get(eout = repcap.Eout.
↪Default)
```

No command help available

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

return
source_list: No help available

6.29.1.2 Source

SCPI Command :

```
TRIGger:BASE:EOUT<n>:SOURce
```

class SourceCls

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

get(*eout=Eout.Default*) → str

```
# SCPI: TRIGger:BASE:EOUT<n>:SOURce
value: str = driver.trigger.eout.source.get(eout = repcap.Eout.Default)
```

No command help available

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

return
source: No help available

set(*source: str*, *eout=Eout.Default*) → None

```
# SCPI: TRIGger:BASE:EOUT<n>:SOURce
driver.trigger.eout.source.set(source = 'abc', eout = repcap.Eout.Default)
```

No command help available

param source
No help available

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

6.29.2 ExtA

SCPI Commands :

```
TRIGger:BASE:EXTA:SOURce
TRIGger:BASE:EXTA:DIRection
TRIGger:BASE:EXTA:SLOPe
```

class ExtACls

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

get_direction() → DirectionIo

```
# SCPI: TRIGger:BASE:EXTA:DIRection
value: enums.DirectionIo = driver.trigger.extA.get_direction()
```

Configures the trigger connectors as input or output connectors.

return
direction: IN | OUT IN: Input connector OUT: Output connector

get_slope() → SignalSlope

```
# SCPI: TRIGger:BASE:EXTA:SLOPe
value: enums.SignalSlope = driver.trigger.extA.get_slope()
```

Specifies whether the rising edge or the falling edge of the trigger pulse is generated at the trigger event. The setting applies to output trigger signals provided at the trigger connectors.

return
slope: REDGe | FEDGe REDGe: Rising edge FEDGe: Falling edge

get_source() → str

```
# SCPI: TRIGger:BASE:EXTA:SOURce
value: str = driver.trigger.extA.get_source()
```

Selects the output trigger signals to be routed to the trigger connectors. A list of all supported values can be retrieved using TRIGger:BASE:EXTA|EXTB:CATalog:SOURce?.

return
source: No help available

set_direction(*direction: DirectionIo*) → None

```
# SCPI: TRIGger:BASE:EXTA:DIRection
driver.trigger.extA.set_direction(direction = enums.DirectionIo.IN)
```

Configures the trigger connectors as input or output connectors.

param direction
IN | OUT IN: Input connector OUT: Output connector

set_slope(*slope: SignalSlope*) → None

```
# SCPI: TRIGger:BASE:EXTA:SLOPe
driver.trigger.extA.set_slope(slope = enums.SignalSlope.FEDGe)
```

Specifies whether the rising edge or the falling edge of the trigger pulse is generated at the trigger event. The setting applies to output trigger signals provided at the trigger connectors.

param slope
REDGe | FEDGe REDGe: Rising edge FEDGe: Falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:BASE:EXTA:SOURce
driver.trigger.extA.set_source(source = 'abc')
```

Selects the output trigger signals to be routed to the trigger connectors. A list of all supported values can be retrieved using TRIGger:BASE:EXTA|EXTB:CATalog:SOURce?.

param source
string

Cloning the Group

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

Subgroups

6.29.2.1 Catalog

SCPI Command :

```
TRIGger:BASE:EXTA:CATalog:SOURce
```

class CatalogCls

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

get_source() → List[str]

```
# SCPI: TRIGger:BASE:EXTA:CATalog:SOURce
value: List[str] = driver.trigger.extA.catalog.get_source()
```

Lists all trigger source values that can be set using TRIGger:BASE:EXTA|EXTB:SOURce.

return
source_list: string Comma-separated list of strings, one string per supported value

6.29.3 ExtB

SCPI Commands :

```
TRIGger:BASE:EXTB:DIRection
TRIGger:BASE:EXTB:SOURce
TRIGger:BASE:EXTB:SLOPe
```

class ExtBCls

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

get_direction() → DirectionIo

```
# SCPI: TRIGger:BASE:EXTB:DIRection
value: enums.DirectionIo = driver.trigger.extB.get_direction()
```

Configures the trigger connectors as input or output connectors.

return
direction: IN | OUT IN: Input connector OUT: Output connector

get_slope() → SignalSlope

```
# SCPI: TRIGger:BASE:EXTB:SLOPe
value: enums.SignalSlope = driver.trigger.extB.get_slope()
```

Specifies whether the rising edge or the falling edge of the trigger pulse is generated at the trigger event. The setting applies to output trigger signals provided at the trigger connectors.

return
slope: REDGe | FEDGe REDGe: Rising edge FEDGe: Falling edge

get_source() → str

```
# SCPI: TRIGger:BASE:EXTB:SOURce
value: str = driver.trigger.extB.get_source()
```

Selects the output trigger signals to be routed to the trigger connectors. A list of all supported values can be retrieved using TRIGger:BASE:EXTA|EXTB:CATalog:SOURce?.

return
source: No help available

set_direction(*direction: DirectionIo*) → None

```
# SCPI: TRIGger:BASE:EXTB:DIRection
driver.trigger.extB.set_direction(direction = enums.DirectionIo.IN)
```

Configures the trigger connectors as input or output connectors.

param direction
IN | OUT IN: Input connector OUT: Output connector

set_slope(*slope: SignalSlope*) → None

```
# SCPI: TRIGger:BASE:EXTB:SLOPe
driver.trigger.extB.set_slope(slope = enums.SignalSlope.FEDGe)
```

Specifies whether the rising edge or the falling edge of the trigger pulse is generated at the trigger event. The setting applies to output trigger signals provided at the trigger connectors.

param slope
REDGe | FEDGe REDGe: Rising edge FEDGe: Falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:BASE:EXTB:SOURce
driver.trigger.extB.set_source(source = 'abc')
```

Selects the output trigger signals to be routed to the trigger connectors. A list of all supported values can be retrieved using TRIGger:BASE:EXTA|EXTB:CATalog:SOURce?.

param source
string

Cloning the Group

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

Subgroups

6.29.3.1 Catalog

SCPI Command :

```
TRIGger:BASE:EXTB:CATalog:SOURce
```

class CatalogCls

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

get_source() → List[str]

```
# SCPI: TRIGger:BASE:EXTB:CATalog:SOURce
value: List[str] = driver.trigger.extB.catalog.get_source()
```

Lists all trigger source values that can be set using TRIGger:BASE:EXTA|EXTB:SOURce.

return
source_list: string Comma-separated list of strings, one string per supported value

6.29.4 Uinitiated<Trigger>

RepCap Settings

```
# Range: Trg1 .. Trg4
rc = driver.trigger.uinitiated.repcap_trigger_get()
driver.trigger.uinitiated.repcap_trigger_set(repcap.Trigger.Trg1)
```

class UinitiatedCls

Uinitiated commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Trigger, default value after init: Trigger.Trg1

Cloning the Group

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

Subgroups

6.29.4.1 Execute

SCPI Command :

```
TRIGger:BASE:UINitiated<n>:EXECute
```

class ExecuteCls

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

set(*trigger=Trigger.Default*) → None

```
# SCPI: TRIGger:BASE:UINitiated<n>:EXECute
driver.trigger.uinitiated.execute.set(trigger = repcap.Trigger.Default)
```

Initiates the generation of a 'User Initiated Trigger' signal.

param trigger
optional repeated capability selector. Default value: Trg1 (settable in the interface 'Uinitiated')

set_with_opc(*trigger=Trigger.Default*, *opc_timeout_ms: int = -1*) → None

6.30 TriggerInvoke

SCPI Command :

```
*TRG
```

class TriggerInvokeCls

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

set() → None

```
# SCPI: *TRG
driver.triggerInvoke.set()
```

No command help available

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

```
# SCPI: *TRG
driver.triggerInvoke.set_with_opc()
```

No command help available

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

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

6.31 Unit

SCPI Commands :

```
UNIT:CONDuctance
UNIT:CHARge
UNIT:CAPacity
UNIT:ENERgy
UNIT:FREQuency
UNIT:RESistor
UNIT:VOLTage
UNIT:ANGLe
UNIT:LENGth
UNIT:CURRent
UNIT:POWer
UNIT:TEMPerature
UNIT:TIME
```

class UnitCls

Unit commands group definition. 13 total commands, 0 Subgroups, 13 group commands

get_angle() → DefaultUnitAngle

```
# SCPI: UNIT:ANGLe
value: enums.DefaultUnitAngle = driver.unit.get_angle()
```

No command help available

return
default_unit_angle: No help available

get_capacity() → DefaultUnitCapacity

```
# SCPI: UNIT:CAPacity
value: enums.DefaultUnitCapacity = driver.unit.get_capacity()
```

No command help available

return
default_unit_capacity: No help available

get_charge() → DefaultUnitCharge

```
# SCPI: UNIT:CHARge
value: enums.DefaultUnitCharge = driver.unit.get_charge()
```

No command help available

return
default_unit_charge: No help available

get_conductance() → DefaultUnitConductance

```
# SCPI: UNIT:CONDuctance
value: enums.DefaultUnitConductance = driver.unit.get_conductance()
```

No command help available

return
default_unit_conductance: No help available

get_current() → DefaultUnitCurrent

```
# SCPI: UNIT:CURRent
value: enums.DefaultUnitCurrent = driver.unit.get_current()
```

No command help available

return
default_unit_current: No help available

get_energy() → DefaultUnitEnergy

```
# SCPI: UNIT:ENERgy
value: enums.DefaultUnitEnergy = driver.unit.get_energy()
```

No command help available

return
default_unit_energy: No help available

get_frequency() → DefaultUnitFrequency

```
# SCPI: UNIT:FREQuency
value: enums.DefaultUnitFrequency = driver.unit.get_frequency()
```

No command help available

return
default_unit_frequency: No help available

get_length() → DefaultUnitLenght

```
# SCPI: UNIT:LENGth
value: enums.DefaultUnitLenght = driver.unit.get_length()
```

No command help available

return
default_unit_lenght: No help available

get_power() → DefaultUnitPower

```
# SCPI: UNIT:POWer
value: enums.DefaultUnitPower = driver.unit.get_power()
```

No command help available

return
default_unit_power: No help available

get_resistor() → DefaultUnitResistor

```
# SCPI: UNIT:RESistor
value: enums.DefaultUnitResistor = driver.unit.get_resistor()
```

No command help available

return
default_unit_resistor: No help available

get_temperature() → DefaultUnitTemperature

```
# SCPI: UNIT:TEMPerature
value: enums.DefaultUnitTemperature = driver.unit.get_temperature()
```

No command help available

return
default_unit_temperature: No help available

get_time() → DefaultUnitTime

```
# SCPI: UNIT:TIME
value: enums.DefaultUnitTime = driver.unit.get_time()
```

No command help available

return
default_unit_time: No help available

get_voltage() → DefaultUnitVoltage

```
# SCPI: UNIT:VOLTage
value: enums.DefaultUnitVoltage = driver.unit.get_voltage()
```

No command help available

return
default_unit_voltage: No help available

set_angle(*default_unit_angle: DefaultUnitAngle*) → None

```
# SCPI: UNIT:ANGLe
driver.unit.set_angle(default_unit_angle = enums.DefaultUnitAngle.DEG)
```

No command help available

param default_unit_angle
No help available

set_capacity(*default_unit_capacity: DefaultUnitCapacity*) → None

```
# SCPI: UNIT:CAPacity
driver.unit.set_capacity(default_unit_capacity = enums.DefaultUnitCapacity.AF)
```

No command help available

param default_unit_capacity
No help available

set_charge(*default_unit_charge: DefaultUnitCharge*) → None

```
# SCPI: UNIT:CHARge
driver.unit.set_charge(default_unit_charge = enums.DefaultUnitCharge.AC)
```

No command help available

param default_unit_charge
No help available

set_conductance(*default_unit_conductance: DefaultUnitConductance*) → None

```
# SCPI: UNIT:CONDuctance
driver.unit.set_conductance(default_unit_conductance = enums.
↪DefaultUnitConductance.ASIE)
```

No command help available

param default_unit_conductance
No help available

set_current(*default_unit_current: DefaultUnitCurrent*) → None

```
# SCPI: UNIT:CURRent
driver.unit.set_current(default_unit_current = enums.DefaultUnitCurrent.A)
```

No command help available

param default_unit_current
No help available

set_energy(*default_unit_energy: DefaultUnitEnergy*) → None

```
# SCPI: UNIT:ENERgy
driver.unit.set_energy(default_unit_energy = enums.DefaultUnitEnergy.AJ)
```

No command help available

param default_unit_energy
No help available

set_frequency(*default_unit_frequency: DefaultUnitFrequency*) → None

```
# SCPI: UNIT:FREQuency
driver.unit.set_frequency(default_unit_frequency = enums.DefaultUnitFrequency.
↪AHZ)
```

No command help available

param default_unit_frequency
No help available

set_length(*default_unit_lenght: DefaultUnitLenght*) → None

```
# SCPI: UNIT:LENGth
driver.unit.set_length(default_unit_lenght = enums.DefaultUnitLenght.AM)
```

No command help available

param default_unit_lenght
No help available

set_power(*default_unit_power: DefaultUnitPower*) → None

```
# SCPI: UNIT:POWer
driver.unit.set_power(default_unit_power = enums.DefaultUnitPower.AW)
```

No command help available

param default_unit_power
No help available

set_resistor(*default_unit_resistor: DefaultUnitResistor*) → None

```
# SCPI: UNIT:RESistor
driver.unit.set_resistor(default_unit_resistor = enums.DefaultUnitResistor.AOHM)
```

No command help available

param default_unit_resistor
No help available

set_temperature(*default_unit_temperature: DefaultUnitTemperature*) → None

```
# SCPI: UNIT:TEMPerature
driver.unit.set_temperature(default_unit_temperature = enums.
↪DefaultUnitTemperature.C)
```

No command help available

param default_unit_temperature
No help available

set_time(*default_unit_time: DefaultUnitTime*) → None

```
# SCPI: UNIT:TIME
driver.unit.set_time(default_unit_time = enums.DefaultUnitTime.AS)
```

No command help available

param default_unit_time
No help available

set_voltage(*default_unit_voltage: DefaultUnitVoltage*) → None

```
# SCPI: UNIT:VOLTage
driver.unit.set_voltage(default_unit_voltage = enums.DefaultUnitVoltage.AV)
```

No command help available

param default_unit_voltage
No help available

6.32 Write

class WriteCls

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

Cloning the Group

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

Subgroups

6.32.1 Eeprom

class EepromCls

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

Cloning the Group

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

Subgroups

6.32.1.1 Data

SCPI Command :

```
WRITe:EEPRom:DATA
```

class DataCls

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

set(*path: str*, *reserve: str*) → None

```
# SCPI: WRITe:EEPRom:DATA
driver.write.eeprom.data.set(path = 'abc', reserve = 'abc')
```

No command help available

param path
No help available

param reserve
No help available

CHAPTER

SEVEN

RSCMWBASE UTILITIES

class Utilities

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

Access snippet: `utils = RsCmwBase.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsCmwBase.utilities.logger`

property driver_version: str

Returns the instrument driver version.

property idn_string: str

Returns instrument's identification string - the response on the SCPI command *IDN?

property manufacturer: str

Returns manufacturer of the instrument.

property full_instrument_model_name: str

Returns the current instrument's full name e.g. 'FSW26'.

property instrument_model_name: str

Returns the current instrument's family name e.g. 'FSW'.

property supported_models: List[str]

Returns a list of the instrument models supported by this instrument driver.

property instrument_firmware_version: str

Returns instrument's firmware version.

property instrument_serial_number: str

Returns instrument's serial_number.

query_opc(*timeout: int = 0*) → int

SCPI command: *OPC? Queries the instrument's OPC bit and hence it waits until the instrument reports operation complete. If you define timeout > 0, the VISA timeout is set to that value just for this method call.

property instrument_status_checking: bool

Sets / returns Instrument Status Checking. When True (default is True), all the driver methods and properties are sending "SYSTem:ERRor?" at the end to immediately react on error that might have occurred. We recommend to keep the state checking ON all the time. Switch it OFF only in rare cases when you require maximum speed. The default state after initializing the session is ON.

property encoding: str

Returns string<=>bytes encoding of the session.

property opc_query_after_write: bool

Sets / returns Instrument *OPC? query sending after each command write. When True, (default is False) the driver sends *OPC? every time a write command is performed. Use this if you want to make sure your sequence is performed command-after-command.

property bin_float_numbers_format: BinFloatFormat

Sets / returns format of float numbers when transferred as binary data.

property bin_int_numbers_format: BinIntFormat

Sets / returns format of integer numbers when transferred as binary data.

clear_status() → None

Clears instrument's status system, the session's I/O buffers and the instrument's error queue.

query_all_errors() → List[str]

Queries and clears all the errors from the instrument's error queue. The method returns list of strings as error messages. If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty. If you want to include the error codes, call the query_all_errors_with_codes()

query_all_errors_with_codes() → List[Tuple[int, str]]

Queries and clears all the errors from the instrument's error queue. The method returns list of tuples (code: int, message: str). If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty.

property instrument_options: List[str]

Returns all the instrument options. The options are sorted in the ascending order starting with K-options and continuing with B-options.

reset() → None

SCPI command: *RST Sends *RST command + calls the clear_status().

default_instrument_setup() → None

Custom steps performed at the init and at the reset().

self_test(*timeout: int = None*) → Tuple[int, str]

SCPI command: *TST? Performs instrument's self-test. Returns tuple (code:int, message: str). Code 0 means the self-test passed. You can define the custom timeout in milliseconds. If you do not define it, the default selftest timeout is used (usually 60 secs).

is_connection_active() → bool

Returns true, if the VISA connection is active and the communication with the instrument still works.

reconnect(*force_close: bool = False*) → bool

If the connection is not active, the method tries to reconnect to the device If the connection is active, and force_close is False, the method does nothing. If the connection is active, and force_close is True, the method closes, and opens the session again. Returns True, if the reconnection has been performed.

property resource_name: int

Returns the resource name used in the constructor

property opc_timeout: int

Sets / returns timeout in milliseconds for all the operations that use OPC synchronization.

property visa_timeout: int

Sets / returns visa IO timeout in milliseconds.

property data_chunk_size: int

Sets / returns the maximum size of one block transferred during write/read operations

property visa_manufacturer: int

Returns the manufacturer of the current VISA session.

process_all_commands() → None

SCPI command: *WAI Stops further commands processing until all commands sent before *WAI have been executed.

write_str(*cmd: str*) → None

Writes the command to the instrument.

write(*cmd: str*) → None

This method is an alias to the write_str(). Writes the command to the instrument as string.

write_int(*cmd: str*, *param: int*) → None

Writes the command to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2'

write_int_with_opc(*cmd: str*, *param: int*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2' If you do not provide timeout, the method uses current opc_timeout.

write_float(*cmd: str*, *param: float*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6'

write_float_with_opc(*cmd: str*, *param: float*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6' If you do not provide timeout, the method uses current opc_timeout.

write_bool(*cmd: str*, *param: bool*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON'

write_bool_with_opc(*cmd: str*, *param: bool*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON' If you do not provide timeout, the method uses current opc_timeout.

query_str(*query: str*) → str

Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query(*query: str*) → str

This method is an alias to the query_str(). Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query_bool(*query: str*) → bool

Sends the query to the instrument and returns the response as boolean.

query_int(*query: str*) → int

Sends the query to the instrument and returns the response as integer.

query_float(*query: str*) → float

Sends the query to the instrument and returns the response as float.

write_str_with_opc(*cmd: str*, *timeout: int = None*) → None

Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

write_with_opc(*cmd: str*, *timeout: int = None*) → None

This method is an alias to the write_str_with_opc(). Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

query_str_with_opc(*query: str*, *timeout: int = None*) → str

Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_with_opc(*query: str*, *timeout: int = None*) → str

This method is an alias to the query_str_with_opc(). Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_bool_with_opc(*query: str*, *timeout: int = None*) → bool

Sends the opc-synced query to the instrument and returns the response as boolean. If you do not provide timeout, the method uses current opc_timeout.

query_int_with_opc(*query: str*, *timeout: int = None*) → int

Sends the opc-synced query to the instrument and returns the response as integer. If you do not provide timeout, the method uses current opc_timeout.

query_float_with_opc(*query: str*, *timeout: int = None*) → float

Sends the opc-synced query to the instrument and returns the response as float. If you do not provide timeout, the method uses current opc_timeout.

write_bin_block(*cmd: str*, *payload: bytes*) → None

Writes all the payload as binary data block to the instrument. The binary data header is added at the beginning of the transmission automatically, do not include it in the payload!!!

query_bin_block(*query: str*) → bytes

Queries binary data block to bytes. Throws an exception if the returned data was not a binary data. Returns data:bytes

query_bin_block_with_opc(*query: str*, *timeout: int = None*) → bytes

Sends a OPC-synced query and returns binary data block to bytes. If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_float_list(*query: str*) → List[float]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_float_list_with_opc(*query: str*, *timeout: int = None*) → List[float]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_int_list(*query: str*) → List[int]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_int_list_with_opc(*query: str*, *timeout: int = None*) → List[int]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_block_to_file(*query: str*, *file_path: str*, *append: bool = False*) → None

Queries binary data block to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data. Example for transferring a file from Instrument -> PC: query = f'MMEM:DATA? '{INSTR_FILE_PATH}'". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

query_bin_block_to_file_with_opc(*query: str*, *file_path: str*, *append: bool = False*, *timeout: int = None*) → None

Sends a OPC-synced query and writes the returned data to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data.

write_bin_block_from_file(*cmd: str*, *file_path: str*) → None

Writes data from the file as binary data block to the instrument using the provided command. Example for transferring a file from PC -> Instrument: cmd = f'MMEM:DATA '{INSTR_FILE_PATH}',". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

send_file_from_pc_to_instrument(*source_pc_file: str*, *target_instr_file: str*) → None

SCPI Command: MMEM:DATA

Sends file from PC to the instrument

read_file_from_instrument_to_pc(*source_instr_file: str*, *target_pc_file: str*, *append_to_pc_file: bool = False*) → None

SCPI Command: MMEM:DATA?

Reads file from instrument to the PC.

Set the `append_to_pc_file` to True if you want to append the read content to the end of the existing PC file

get_last_sent_cmd() → str

Returns the last commands sent to the instrument. Only works in simulation mode

go_to_local() → None

Puts the instrument into local state.

go_to_remote() → None

Puts the instrument into remote state.

get_lock() → RLock

Returns the thread lock for the current session.

By default:

- If you create standard new RsCmwBase instance with new VISA session, the session gets a new thread lock. You can assign it to other RsCmwBase sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsCmwBase 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

RSCMWBASE LOGGER

Check the usage in the Getting Started chapter *here*.

class ScpiLogger

Base class for SCPI logging

mode

Sets the logging ON or OFF. Additionally, you can set the logging ON only for errors. Possible values:

- LoggingMode.Off - logging is switched OFF
- LoggingMode.On - logging is switched ON
- LoggingMode.Errors - logging is switched ON, but only for error entries
- LoggingMode.Default - sets the logging to default - the value you have set with logger.default_mode

default_mode

Sets / returns the default logging mode. You can recall the default mode by calling the logger.mode = LoggingMode.Default.

Data Type
LoggingMode

device_name: str

Use this property to change the resource name in the log from the default Resource Name (e.g. TCPIP::192.168.2.101::INSTR) to another name e.g. 'MySigGen1'.

set_logging_target(*target*, *console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target - the target must implement write() and flush(). You can optionally set the console and UDP logging ON or OFF. This method switches the logging target global OFF.

get_logging_target()

Based on the global_mode, it returns the logging target: either the local or the global one.

set_logging_target_global(*console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target to global. The global target must be defined. You can optionally set the console and UDP logging ON or OFF.

log_to_console

Returns logging to console status.

log_to_udp

Returns logging to UDP status.

log_to_console_and_udp

Returns true, if both logging to UDP and console in are True.

info_raw(*log_entry: str*, *add_new_line: bool = True*) → None

Method for logging the raw string without any formatting.

info(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one info entry. For binary log_string, use the info_bin()

error(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one error entry.

set_relative_timestamp(*timestamp: datetime*) → None

If set, the further timestamps will be relative to the entered time.

set_relative_timestamp_now() → None

Sets the relative timestamp to the current time.

get_relative_timestamp() → datetime

Based on the global_mode, it returns the relative timestamp: either the local or the global one.

clear_relative_timestamp() → None

Clears the reference time, and the further logging continues with absolute times.

flush() → None

Flush all the entries.

log_status_check_ok

Sets / returns the current status of status checking OK. If True (default), the log contains logging of the status checking 'Status check: OK'. If False, the 'Status check: OK' is skipped - the log is more compact. Errors will still be logged.

clear_cached_entries() → None

Clears potential cached log entries. Cached log entries are generated when the Logging is ON, but no target has been defined yet.

set_format_string(*value: str*, *line_divider: str = '\n'*) → None

Sets new format string and line divider. If you just want to set the line divider, set the format string value=None The original format string is: `PAD_LEFT12(%START_TIME%) PAD_LEFT25(%DEVICE_NAME%) PAD_LEFT12(%DURATION%) %LOG_STRING_INFO%: %LOG_STRING%`

restore_format_string() → None

Restores the original format string and the line divider to LF

abbreviated_max_len_ascii: int

Defines the maximum length of one ASCII log entry. Default value is 200 characters.

abbreviated_max_len_bin: int

Defines the maximum length of one Binary log entry. Default value is 2048 bytes.

abbreviated_max_len_list: int

Defines the maximum length of one list entry. Default value is 100 elements.

bin_line_block_size: int

Defines number of bytes to display in one line. Default value is 16 bytes.

udp_port

Returns udp logging port.

target_auto_flushing

Returns status of the auto-flushing for the logging target.

CHAPTER

NINE

RSCMWBASE EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

TEN

INDEX

INDEX

Symbols

A

B

C

D

E

T

U

W