RsCMPX_Gprf

Release 5.0.91.48

Rohde & Schwarz

Apr 25, 2024

CONTENTS:

CONTENTS:

CHAPTER

ONE

REVISION HISTORY

1.1 RsCMPX_Gprf

Rohde & Schwarz CMX/CMP/PVT Global Purpose RF RsCMPX_Gprf instrument driver.

Basic Hello-World code:

```
from RsCMPX_Gprf import *

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

Supported instruments: CMX500, CMP200, CMP180, PVT360

The package is hosted here: https://pypi.org/project/RsCMPX-Gprf/

Documentation: https://RsCMPX-Gprf.readthedocs.io/

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

1.1.1 Version history

Latest release notes summary: Added missing measurement SCPI from MMI sub-system

Version 5.0.91

- Added missing measurement SCPI from MMI sub-system

Version 5.0.90

- Update for CMP FW 5.0.90

Version 5.0.20

- Update for CMP200 5.0.20

Version 4.0.172

- Fixed documentation

Version 4.0.171

- Updated MMI commands for the FW 4.0.170

Version 4.0.170

- Fixed documentation

Version 4.0.140

- Update of RsCMPX_Gprf to FW 4.0.140 from the complete FW package 7.10.0

Version 4.0.110

- Fixed generation of specific interfaces, new core

Version 4.0.20

- Update for the FW verision 4.0.20

Version 4.0.12

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsCMPX_Gprf 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 example for RsCmpx-Base and RsCmpx-Gprf:

```
"""
# GitHub examples repository path: CMXP/Python/RsCmxp_xxx_ScpiPackages

Example on how to use the python RsCmx auto-generated instrument drivers for
RsCmpx_Base and RsCmpx_Gprf (Base and GPRF) in one script with shared VISA session.
"""

from RsCMPX_Base.RsCMPX_Base import RsCMPX_Base  # install from pypi.org
from RsCMPX_Base import enums as base_enums
from RsCMPX_Base import repcap as base_repcap

from RsCMPX_Gprf.RsCMPX_Gprf import RsCMPX_Gprf  # install from pypi.org
from RsCMPX_Gprf.CustomFiles.reliability import ReliabilityEventArgs
from RsCMPX_Gprf import enums as gprf_enums
from RsCMPX_Gprf import repcap as gprf_repcaps
```

(continues on next page)

(continued from previous page)

```
# CMX Base init
cmx_base = RsCMPX_Base('TCPIP::10.112.1.116', False, True)
print(f'CMX Base IND: {cmx_base.utilities.idn_string}')
print(f'CMX Instrument options:\n{",".join(cmx_base.utilities.instrument_options)}')
cmx_base.utilities.visa_timeout = 5000  # default is 10000

# Sends OPC after each command
cmx_base.utilities.opc_query_after_write = False
# Checks for syst:err? after each command / query - default value after init is True
cmx_base.utilities.instrument_status_checking = True

# Self-test
self_test = cmx_base.utilities.self_test()
print(f'CMW self-test result: {self_test} - {"Passed" if self_test[0] == 0 else "Failed"}
↪"')
# Reference Frequency Source
cmx_base.system.reference.frequency.source_set(base_enums.SourceIntExt.INTernal)

# CMX RsCMPX_Gprf Init - reuse the session of the cmx_base, rather than creating another␣
↪one
cmx_gprf = RsCMPX_Gprf.from_existing_session(cmx_base)
cmx_gprf.utilities.visa_timeout = 5000

# Driver's Interface reliability offers a convenient way of reacting on the return value␣
↪Reliability Indicator
cmx_gprf.reliability.ExceptionOnError = True  # default is 10000


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


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

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

# Close the sessions
cmx_gprf.close()
cmx_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

RsCMPX_Gprf 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 RsCMPX_Gprf`

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 `RsCMPX_Gprf` 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 RsCMPX_Gprf offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCMPX_Gprf needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCMPX_Gprf package to your computer from the pypi.org: https://pypi.org/project/RsCMPX_Gprf/#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\RsCMPX_Gprf-5.0.91.48.tar`

2.3 Finding Available Instruments

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

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

from RsCMPX_Gprf import *

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

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

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

- Installed RsCMPX_Gprf Python module Version 5.0.91 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCMPX_Gprf import *

# A good practice is to assure that you have a certain minimum version installed
RsCMPX_Gprf.assert_minimum_version('5.0.91')
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 = RsCMPX_Gprf(resource_string_1)

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

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

from RsCMPX_Gprf import *

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

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

(continues on next page)

(continued from previous page)

```
from RsCMPX_Gprf import *

driver = RsCMPX_Gprf('TCPIP::192.168.56.101::5025::SOCKET', True, True, "SelectVisa=
↪'socket'")
print(f'Visa manufacturer: {driver.utilities.visa_manufacturer}')
print(f"\nHello, I am: '{driver.utilities.idn_string}'")

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

Warning: Not using VISA can cause problems by debugging when you want to use the communication Trace Tool. The good news is, you can easily switch to use VISA and back just by changing the constructor arguments. The rest of your code stays unchanged.

Simulating Session

If a colleague is currently occupying your instrument, leave him in peace, and open a simulating session:

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

More `option_string` tokens are separated by comma:

```
driver = RsCMPX_Gprf('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 RsCMPX_Gprf objects:

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

from RsCMPX_Gprf import *

driver1 = RsCMPX_Gprf('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsCMPX_Gprf.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}')
```

(continues on next page)

(continued from previous page)

```
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 RsCMPX_Gprf 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 RsCMPX_Gprf import *

driver = RsCMPX_Gprf('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 RsCMPX_Gprf import *

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

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

One additional feature we need to mention here: **VISA timeout**. To simplify, VISA timeout plays a role in each `query_xxx()`, where the controller (your PC) has to prevent waiting forever for an answer from your instrument. VISA timeout defines that maximum waiting time. You can set/read it with the `visa_timeout` property:

```
# Timeout in milliseconds
driver.utilities.visa_timeout = 3000
```

After this time, the RsCMPX_Gprf raises an exception. Speaking of exceptions, an important feature of the RsCMPX_Gprf 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 RsCMPX_Gprf import *

driver = RsCMPX_Gprf('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

RsCMPX_Gprf 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 RsCMPX_Gprf 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 RsCMPX_Gprf import *

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

RsCMPX_Gprf 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 RsCMPX_Gprf 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 = RsCMPX_Gprf('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
```

(continues on next page)

(continued from previous page)

```
# 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 RsCMPX_Gprf 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, RsCMPX_Gprf 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 RsCMPX_Gprf object
"""

import threading
from RsCMPX_Gprf import *
```

(continues on next page)

(continued from previous page)

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


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

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsCMPX_Gprf import *


def execute(session: RsCMPX_Gprf, 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 = RsCMPX_Gprf('TCPIP::192.168.56.101::INSTR')
driver2 = RsCMPX_Gprf.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 = []
```

(continues on next page)

(continued from previous page)

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

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

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

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

Run the following example:

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

import threading
from RsCMPX_Gprf import *


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

(continues on next page)

(continued from previous page)

```
driver2.utilities.visa_timeout = 200

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

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

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

(continues on next page)

(continued from previous page)

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

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

Console output:

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

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

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

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

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

See the full logger help *here*.

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

```
driver = RsCMPX_Gprf('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 RsCMPX_Gprf import *

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

(continues on next page)

(continued from previous page)

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

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

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

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

driver.utilities.reset()

# Close the session
driver.close()

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

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
from RsCMPX_Gprf import *

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

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

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

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

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

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

Console output:

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

Notice the following:

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

CHAPTER

THREE

ENUMS

3.1 All

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

3.2 Amplification

```
# Example value:
value = enums.Amplification.HIGH
# All values (4x):
HIGH | LOW | MAXimum | MEDium
```

3.3 ArbFile

```
# Example value:
value = enums.ArbFile.ABSPath
# All values (3x):
ABSPath | DEF | TRUTh
```

3.4 ArbSegmentsMode

```
# Example value:
value = enums.ArbSegmentsMode.AUTO
# All values (3x):
AUTO | CONTinuous | CSEamless
```

3.5 AveragingMode

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

3.6 BasebandMode

```
# Example value:
value = enums.BasebandMode.ARB
# All values (3x):
ARB | CW | DTONe
```

3.7 CcdfMode

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

3.8 ConnectionSource

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

3.9 Detector

```
# Example value:
value = enums.Detector.AUTopeak
# All values (6x):
AUTopeak | AVERage | MAXPeak | MINPeak | RMS | SAMPle
```

3.10 DetectorBasic

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

3.11 DeviceMode

```
# Example value:
value = enums.DeviceMode.M2X2
# All values (3x):
M2X2 | M4X4 | NONE
```

3.12 DeviceType

```
# Example value:
value = enums.DeviceType.NONE
# All values (2x):
NONE | Z24
```

3.13 DigitalFilterType

```
# Example value:
value = enums.DigitalFilterType.BANDpass
# All values (5x):
BANDpass | CDMA | GAUSs | TDSCdma | WCDMa
```

3.14 ExtPwrSensorAvgMode

```
# Example value:
value = enums.ExtPwrSensorAvgMode.MANual
# All values (3x):
MANual | NSR | RES
```

3.15 FileSave

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

3.16 FilterCriteria

```
# Example value:
value = enums.FilterCriteria.LOSupport
# All values (1x):
LOSupport
```

3.17 FilterType

```
# Example value:
value = enums.FilterType.B10Mhz
# All values (5x):
B10Mhz | B1MHz | GAUSs | NY1Mhz | NYQuist
```

3.18 GeneratorState

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

3.19 IncTransition

```
# Example value:
value = enums.IncTransition.IMMediate
# All values (6x):
IMMediate | RMARker | WMA1 | WMA2 | WMA3 | WMA4
```

3.20 InstrumentType

```
# Example value:
value = enums.InstrumentType.PROTocol
# All values (2x):
PROTocol | SIGNaling
```

3.21 IqFormat

```
# Example value:
value = enums.IqFormat.IQ
# All values (2x):
IQ | RPHI
```

3.22 IqRecBypass

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

3.23 ListIncrement

```
# Example value:
value = enums.ListIncrement.ACYCles
# All values (5x):
ACYCles | DTIMe | MEASurement | TRIGger | USER
```

3.24 ListSubMode

```
# Example value:
value = enums.ListSubMode.AUTO
# All values (6x):
AUTO | BBGenerator | BBMeasurement | OTHer | SINGle | STEP
```

3.25 LoLevel

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

3.26 LowHigh

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

3.27 MagnitudeUnit

```
# Example value:
value = enums.MagnitudeUnit.RAW
# All values (2x):
RAW | VOLT
```

3.28 MeasMode

```
# Example value:
value = enums.MeasMode.CCALibration
# All values (3x):
CCALibration | OOPen | OSHort
```

3.29 MeasScenario

```
# Example value:
value = enums.MeasScenario.CSPath
# All values (5x):
CSPath | MAIQ | MAPR | SALone | UNDefined
```

3.30 MeasurementMode

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

3.31 NameStyle

```
# Example value:
value = enums.NameStyle.FQName
# All values (3x):
FQName | LNAMe | NAME
```

3.32 OffsetMode

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

3.33 ParameterSetMode

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

3.34 PathLossState

```
# Example value:
value = enums.PathLossState.NCAP
# All values (3x):
NCAP | PEND | RDY
```

3.35 PwrSensorResolution

```
# Example value:
value = enums.PwrSensorResolution.PD0
# All values (4x):
PD0 | PD1 | PD2 | PD3
```

3.36 Range

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

3.37 RbwFilterType

```
# Example value:
value = enums.RbwFilterType.BANDpass
# All values (2x):
BANDpass | GAUSs
```

3.38 Repeat

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

3.39 RepeatMode

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

3.40 ResourceState

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

3.41 ResultStatus2

```
# First value:
value = enums.ResultStatus2.DC
# Last value:
value = enums.ResultStatus2.ULEU
# All values (10x):
DC | INV | NAV | NCAP | OFF | OFL | OK | UFL
ULEL | ULEU
```

3.42 RfConnector

```
# First value:
value = enums.RfConnector.I11I
# Last value:
value = enums.RfConnector.RH8
# All values (163x):
I11I | I13I | I15I | I17I | I21I | I23I | I25I | I27I
I31I | I33I | I35I | I37I | I41I | I43I | I45I | I47I
IFI1 | IFI2 | IFI3 | IFI4 | IFI5 | IFI6 | IQ1I | IQ3I
IQ5I | IQ7I | R10D | R11 | R11C | R11D | R12 | R12C
R12D | R12I | R13 | R13C | R14 | R14C | R14I | R15
R16 | R17 | R18 | R21 | R21C | R22 | R22C | R22I
R23 | R23C | R24 | R24C | R24I | R25 | R26 | R27
R28 | R31 | R31C | R32 | R32C | R32I | R33 | R33C
R34 | R34C | R34I | R35 | R36 | R37 | R38 | R41
R41C | R42 | R42C | R42I | R43 | R43C | R44 | R44C
R44I | R45 | R46 | R47 | R48 | RA1 | RA2 | RA3
RA4 | RA5 | RA6 | RA7 | RA8 | RB1 | RB2 | RB3
RB4 | RB5 | RB6 | RB7 | RB8 | RC1 | RC2 | RC3
RC4 | RC5 | RC6 | RC7 | RC8 | RD1 | RD2 | RD3
RD4 | RD5 | RD6 | RD7 | RD8 | RE1 | RE2 | RE3
RE4 | RE5 | RE6 | RE7 | RE8 | RF1 | RF1C | RF2
RF2C | RF2I | RF3 | RF3C | RF4 | RF4C | RF4I | RF5
RF5C | RF6 | RF6C | RF7 | RF7C | RF8 | RF8C | RF9C
RFAC | RFBC | RFBI | RG1 | RG2 | RG3 | RG4 | RG5
RG6 | RG7 | RG8 | RH1 | RH2 | RH3 | RH4 | RH5
RH6 | RH7 | RH8
```

3.43 SelectMode

```
# Example value:
value = enums.SelectMode.EVAL
# All values (4x):
EVAL | MATCh | OPEN | SHORt
```

3.44 SignalDirection

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

3.45 SignalSlope

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

3.46 SignalSlopeExt

```
# Example value:
value = enums.SignalSlopeExt.FALLing
# All values (4x):
FALLing | FEDGe | REDGe | RISing
```

3.47 SourceInt

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

3.48 SpanMode

```
# Example value:
value = enums.SpanMode.FSWeep
# All values (2x):
FSWeep | ZSPan
```

3.49 Statistic

```
# Example value:
value = enums.Statistic.AVERage
# All values (4x):
AVERage | CURRent | MAXimum | MINimum
```

3.50 TargetStateA

```
# Example value:
value = enums.TargetStateA.OFF
# All values (3x):
OFF | RDY | RUN
```

3.51 TargetStateB

```
# Example value:
value = enums.TargetStateB.OFF
# All values (3x):
OFF | RUN | STOP
```

3.52 TargetSyncState

```
# Example value:
value = enums.TargetSyncState.ADJusted
# All values (2x):
ADJusted | PENDing
```

3.53 TimeSource

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

3.54 Timing

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

3.55 TransferMode

```
# Example value:
value = enums.TransferMode.ENABlemode
# All values (2x):
ENABlemode | REQuestmode
```

3.56 Trigger

```
# Example value:
value = enums.Trigger.CLEarlist
# All values (3x):
CLEarlist | GENeratelist | OFF
```

3.57 TriggerPowerMode

```
# Example value:
value = enums.TriggerPowerMode.ALL
# All values (4x):
ALL | ONCE | PRESelect | SWEep
```

3.58 TriggerSequenceMode

```
# Example value:
value = enums.TriggerSequenceMode.ONCE
# All values (2x):
ONCE | PRESelect
```

3.59 TriggerSource

```
# Example value:
value = enums.TriggerSource.EXTernal
# All values (4x):
EXTernal | FREerun | IF | IFPower
```

3.60 TxConnector

```
# First value:
value = enums.TxConnector.I120
# Last value:
value = enums.TxConnector.RH18
# All values (86x):
I120 | I140 | I160 | I180 | I220 | I240 | I260 | I280
I320 | I340 | I360 | I380 | I420 | I440 | I460 | I480
IF01 | IF02 | IF03 | IF04 | IF05 | IF06 | IQ20 | IQ40
IQ60 | IQ80 | R10D | R118 | R1183 | R1184 | R11C | R11D
R110 | R1103 | R1104 | R12C | R12D | R13C | R130 | R14C
R214 | R218 | R21C | R210 | R22C | R23C | R230 | R24C
R258 | R318 | R31C | R310 | R32C | R33C | R330 | R34C
R418 | R41C | R410 | R42C | R43C | R430 | R44C | RA18
RB14 | RB18 | RC18 | RD18 | RE18 | RF18 | RF1C | RF10
RF2C | RF3C | RF30 | RF4C | RF5C | RF6C | RF7C | RF8C
RF9C | RFAC | RFA0 | RFBC | RG18 | RH18
```

3.61 TxiMode

```
# Example value:
value = enums.TxiMode.IREPetition
# All values (2x):
IREPetition | LENTry
```

3.62 UserDebugMode

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

3.63 YesNoStatus

```
# Example value:
value = enums.YesNoStatus.NO
# All values (2x):
NO | YES
```

3.64 ZeroingState

```
# Example value:
value = enums.ZeroingState.FAILed
# All values (2x):
FAILed | PASSed
```

CHAPTER

FOUR

REPCAPS

4.1 Instance (Global)

```
# Setting:
driver.repcap_instance_set(repcap.Instance.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
```

4.2 Bench

```
# First value:
value = repcap.Bench.Nr1
# Range:
Nr1 .. Nr20
# All values (20x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
Nr17 | Nr18 | Nr19 | Nr20
```

4.3 Box

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

4.4 FrequencySource

```
# First value:
value = repcap.FrequencySource.Src1
# Values (2x):
Src1 | Src2
```

4.5 Index

```
# First value:
value = repcap.Index.Ix1
# Range:
Ix1 .. Ix32
# All values (32x):
Ix1 | Ix2 | Ix3 | Ix4 | Ix5 | Ix6 | Ix7 | Ix8
Ix9 | Ix10 | Ix11 | Ix12 | Ix13 | Ix14 | Ix15 | Ix16
Ix17 | Ix18 | Ix19 | Ix20 | Ix21 | Ix22 | Ix23 | Ix24
Ix25 | Ix26 | Ix27 | Ix28 | Ix29 | Ix30 | Ix31 | Ix32
```

4.6 LevelSource

```
# First value:
value = repcap.LevelSource.Src1
# Values (2x):
Src1 | Src2
```

4.7 Marker

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

4.8 Positioner

```
# First value:
value = repcap.Positioner.Ix1
# Range:
Ix1 .. Ix32
# All values (32x):
Ix1 | Ix2 | Ix3 | Ix4 | Ix5 | Ix6 | Ix7 | Ix8
Ix9 | Ix10 | Ix11 | Ix12 | Ix13 | Ix14 | Ix15 | Ix16
```

(continues on next page)

(continued from previous page)

```
Ix17 | Ix18 | Ix19 | Ix20 | Ix21 | Ix22 | Ix23 | Ix24
Ix25 | Ix26 | Ix27 | Ix28 | Ix29 | Ix30 | Ix31 | Ix32
```

4.9 Sensor

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

4.10 Stream

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

CHAPTER

FIVE

EXAMPLES

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

```
"""
# GitHub examples repository path: CMXP/Python/RsCmxp_xxx_ScpiPackages

Example on how to use the python RsCmx auto-generated instrument drivers for
RsCmpx_Base and RsCmpx_Gprf (Base and GPRF) in one script with shared VISA session.
"""

from RsCMPX_Base.RsCMPX_Base import RsCMPX_Base  # install from pypi.org
from RsCMPX_Base import enums as base_enums
from RsCMPX_Base import repcap as base_repcap

from RsCMPX_Gprf.RsCMPX_Gprf import RsCMPX_Gprf  # install from pypi.org
from RsCMPX_Gprf.CustomFiles.reliability import ReliabilityEventArgs
from RsCMPX_Gprf import enums as gprf_enums
from RsCMPX_Gprf import repcap as gprf_repcaps


# CMX Base init
cmx_base = RsCMPX_Base('TCPIP::10.112.1.116', False, True)
print(f'CMX Base IND: {cmx_base.utilities.idn_string}')
print(f'CMX Instrument options:\n{",".join(cmx_base.utilities.instrument_options)}')
cmx_base.utilities.visa_timeout = 5000  # default is 10000

# Sends OPC after each command
cmx_base.utilities.opc_query_after_write = False
# Checks for syst:err? after each command / query - default value after init is True
cmx_base.utilities.instrument_status_checking = True

# Self-test
self_test = cmx_base.utilities.self_test()
print(f'CMW self-test result: {self_test} - {"Passed" if self_test[0] == 0 else "Failed"}
↪"')
# Reference Frequency Source
cmx_base.system.reference.frequency.source_set(base_enums.SourceIntExt.INTernal)

# CMX RsCMPX_Gprf Init - reuse the session of the cmx_base, rather than creating another␣
↪one
cmx_gprf = RsCMPX_Gprf.from_existing_session(cmx_base)
```

(continues on next page)

(continued from previous page)

```
cmx_gprf.utilities.visa_timeout = 5000

# Driver's Interface reliability offers a convenient way of reacting on the return value␣
↪Reliability Indicator
cmx_gprf.reliability.ExceptionOnError = True  # default is 10000


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


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

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

# Close the sessions
cmx_gprf.close()
cmx_base.close()
```

CHAPTER

SIX

RSCMPX_GPRF API STRUCTURE

Global RepCaps

```
driver = RsCMPX_Gprf('TCPIP::192.168.2.101::hislip0')
# Instance range: Inst1 .. Inst32
rc = driver.repcap_instance_get()
driver.repcap_instance_set(repcap.Instance.Inst1)
```

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

859 total commands, 16 Subgroups, 0 group commands

Initializes new RsCMPX_Gprf 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 RsCMPX_Gprf.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 RsCMPX_Gprf session.

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

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

Parameters

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

classmethod get_global_logging_relative_timestamp() → datetime

Returns global common relative timestamp for log entries.

classmethod get_global_logging_target()

Returns global common target stream.

get_session_handle() → object

Returns the underlying session handle.

get_total_execution_time() → timedelta

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

get_total_time() → timedelta

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

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

Finds all the resources defined by the expression

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

Parameters

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

reset_time_statistics() → None

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

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

classmethod set_global_logging_relative_timestamp(*timestamp: datetime*) → None

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

classmethod set_global_logging_relative_timestamp_now() → None

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

classmethod set_global_logging_target(*target*) → None

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

Subgroups

6.1 Add

class AddCls

Add 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.add.clone()
```

Subgroups

6.1.1 System

class SystemCls

System 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.add.system.clone()
```

Subgroups

6.1.1.1 Attenuation

class AttenuationCls

Attenuation 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.add.system.attenuation.clone()
```

Subgroups

6.1.1.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.add.system.attenuation.correctionTable.clone()
```

Subgroups

6.1.1.1.1.1 Globale

SCPI Command :

```
ADD:SYSTem:ATTenuation:CTABle:GLOBal
```

class GlobaleCls

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

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

```
# SCPI: ADD:SYSTem:ATTenuation:CTABle:GLOBal
driver.add.system.attenuation.correctionTable.globale.set(name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3], attenuation = [1.1, 2.2, 3.3])
```

No command help available

param name
No help available

param frequency
No help available

param attenuation
No help available

6.1.1.1.2 Tenvironment

SCPI Command :

```
ADD:SYSTem:ATTenuation:CTABle[:TENVironment]
```

class TenvironmentCls

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

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

```
# SCPI: ADD:SYSTem:ATTenuation:CTABle[:TENVironment]
driver.add.system.attenuation.correctionTable.tenvironment.set(name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3], attenuation = [1.1, 2.2, 3.3])
```

No command help available

param name
No help available

param frequency
No help available

param attenuation
No help available

6.1.2 Tenvironment

class TenvironmentCls

Tenvironment 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.add.tenvironment.clone()
```

Subgroups

6.1.2.1 Spath

class SpathCls

Spath 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.add.tenvironment.spath.clone()
```

Subgroups

6.1.2.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.add.tenvironment.spath.correctionTable.clone()
```

Subgroups

6.1.2.1.1.1 Rx

SCPI Command :

```
ADD:TENVironment:SPATh:CTABle:RX
```

class RxCls

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

set(*name_signal_path: str*, *correction_table: List[str]*) → None

```
# SCPI: ADD:TENVironment:SPATh:CTABle:RX
driver.add.tenvironment.spath.correctionTable.rx.set(name_signal_path = 'abc',␣
↪correction_table = ['abc1', 'abc2', 'abc3'])
```

No command help available

param name_signal_path
No help available

param correction_table
No help available

6.1.2.1.1.2 Tx

SCPI Command :

```
ADD:TENVironment:SPATh:CTABle:TX
```

class TxCls

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

set(*name_signal_path: str*, *correction_table: List[str]*) → None

```
# SCPI: ADD:TENVironment:SPATh:CTABle:TX
driver.add.tenvironment.spath.correctionTable.tx.set(name_signal_path = 'abc',␣
↪correction_table = ['abc1', 'abc2', 'abc3'])
```

No command help available

param name_signal_path
No help available

param correction_table
No help available

6.2 Calibration

class CalibrationCls

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

Cloning the Group

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

Subgroups

6.2.1 Gprf

class GprfCls

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

Cloning the Group

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

Subgroups

6.2.1.1 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.calibration.gprf.measurement.clone()
```

Subgroups

6.2.1.1.1 ExtPwrSensor

class ExtPwrSensorCls

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

Cloning the Group

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

Subgroups

6.2.1.1.1.1 Zero

SCPI Command :

```
CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
```

class ZeroCls

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

get() → ZeroingState

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
value: enums.ZeroingState = driver.calibration.gprf.measurement.extPwrSensor.
↪zero.get()
```

Initiates zeroing of the power sensor or reads the zeroing state. A running external power sensor measurement is interrupted and restarted after the zeroing procedure has been completed. Zeroing takes a few seconds (3 to 10) .

Suppressed linked return values: reliability

return
zeroing_state: 'PASSed': The previous zeroing was successful. 'FAILed': The previous zeroing resulted in an error, e.g. because the signal power was not switched off.

set() → None

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
driver.calibration.gprf.measurement.extPwrSensor.zero.set()
```

Initiates zeroing of the power sensor or reads the zeroing state. A running external power sensor measurement is interrupted and restarted after the zeroing procedure has been completed. Zeroing takes a few seconds (3 to 10) .

Suppressed linked return values: reliability

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

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
driver.calibration.gprf.measurement.extPwrSensor.zero.set_with_opc()
```

Initiates zeroing of the power sensor or reads the zeroing state. A running external power sensor measurement is interrupted and restarted after the zeroing procedure has been completed. Zeroing takes a few seconds (3 to 10) .

Suppressed linked return values: reliability

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

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

6.3 Catalog

class CatalogCls

Catalog commands group definition. 26 total commands, 16 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 Bluetooth

class BluetoothCls

Bluetooth 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.catalog.bluetooth.clone()
```

Subgroups

6.3.1.1 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.catalog.bluetooth.measurement.clone()
```

Subgroups

6.3.1.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.bluetooth.measurement.spath.repcap_stream_get()
driver.catalog.bluetooth.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:BLUetooth:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:BLUetooth:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.bluetooth.measurement.spath.get(stream =␣
↪repcap.Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.2 Cdma

class CdmaCls

Cdma 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.catalog.cdma.clone()
```

Subgroups

6.3.2.1 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.catalog.cdma.measurement.clone()
```

Subgroups

6.3.2.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.cdma.measurement.spath.repcap_stream_get()
driver.catalog.cdma.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:CDMA:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:CDMA:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.cdma.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.3 Gprf

class GprfCls

Gprf 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.catalog.gprf.clone()
```

Subgroups

6.3.3.1 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.catalog.gprf.generator.clone()
```

Subgroups

6.3.3.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.gprf.generator.spath.repcap_stream_get()
driver.catalog.gprf.generator.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:GPRF:GENerator<Instance>:SPATh<StreamNumber>
```

class SpathCls

Spath commands group definition. 3 total commands, 1 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.Nr1

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:GPRF:GENerator<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.gprf.generator.spath.get(stream = repcap.
↪Stream.Default)
```

Returns the names of the available RF connections (broadcast off) or connector groups (broadcast on) .

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

return
name_signal_path: Comma-separated list of strings, one string per name.

Cloning the Group

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

Subgroups

6.3.3.1.1.1 Group

SCPI Commands :

```
CATalog:GPRF:GENerator<Instance>:SPATh:GROup:CONNector
CATalog:GPRF:GENerator<Instance>:SPATh:GROup
```

class GroupCls

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

get(*connector_name: str*) → List[str]

```
# SCPI: CATalog:GPRF:GENerator<Instance>:SPATh:GROup
value: List[str] = driver.catalog.gprf.generator.spath.group.get(connector_name␣
↪= 'abc')
```

Returns the names of the RF connections that start at the specified connector. This command is only relevant if you create your own RF connections (see base manual) . The default RF connections have the same name as the assigned connector.

param connector_name
Start point of the RF connections.

return
signal_path: Comma-separated list of strings, one string per RF connection.

get_connector() → str

```
# SCPI: CATalog:GPRF:GENerator<Instance>:SPATh:GROup:CONNector
value: str = driver.catalog.gprf.generator.spath.group.get_connector()
```

Returns the names of the connectors of the active connector group. Select the connector group via method RsCMPX_Gprf. Route.Gprf.Generator.Spath.value.

return
connector_name: Comma-separated list of values, one value per connector.

6.3.3.2 Measurement

class MeasurementCls

Measurement 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.catalog.gprf.measurement.clone()
```

Subgroups

6.3.3.2.1 Ploss

SCPI Command :

```
CATalog:GPRF:MEASurement<instance>:PLOSs:CNAMe
```

class PlossCls

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

get_cname() → str

```
# SCPI: CATalog:GPRF:MEASurement<instance>:PLOSs:CNAMe
value: str = driver.catalog.gprf.measurement.ploss.get_cname()
```

Returns the names of the available RF connections.

return
connection_names: Comma-separated list of strings, one string per RF connection

6.3.3.2.2 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.gprf.measurement.spath.repcap_stream_get()
driver.catalog.gprf.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:GPRF:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:GPRF:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.gprf.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

Returns the names of the available RF connections.

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

return
name_signal_path: Comma-separated list of strings, one string per RF connection.

Cloning the Group

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

6.3.4 Gsm

class GsmCls

Gsm 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.catalog.gsm.clone()
```

Subgroups

6.3.4.1 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.catalog.gsm.measurement.clone()
```

Subgroups

6.3.4.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.gsm.measurement.spath.repcap_stream_get()
driver.catalog.gsm.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:GSM:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:GSM:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.gsm.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.5 Lte

class LteCls

Lte 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.catalog.lte.clone()
```

Subgroups

6.3.5.1 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.catalog.lte.measurement.clone()
```

Subgroups

6.3.5.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.lte.measurement.spath.repcap_stream_get()
driver.catalog.lte.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:LTE:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:LTE:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.lte.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.6 LteDl

class LteDlCls

LteDl 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.catalog.lteDl.clone()
```

Subgroups

6.3.6.1 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.catalog.lteDl.measurement.clone()
```

Subgroups

6.3.6.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.lteDl.measurement.spath.repcap_stream_get()
driver.catalog.lteDl.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:LTEDl:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:LTEDl:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.lteDl.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.7 Niot

class NiotCls

Niot 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.catalog.niot.clone()
```

Subgroups

6.3.7.1 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.catalog.niot.measurement.clone()
```

Subgroups

6.3.7.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.niot.measurement.spath.repcap_stream_get()
driver.catalog.niot.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:NIOT:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:NIOT:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.niot.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.8 NrDl

class NrDlCls

NrDl 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.catalog.nrDl.clone()
```

Subgroups

6.3.8.1 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.catalog.nrDl.measurement.clone()
```

Subgroups

6.3.8.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.nrDl.measurement.spath.repcap_stream_get()
driver.catalog.nrDl.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:NRDL:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:NRDL:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.nrDl.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.9 NrMmw

class NrMmwCls

NrMmw 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.catalog.nrMmw.clone()
```

Subgroups

6.3.9.1 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.catalog.nrMmw.measurement.clone()
```

Subgroups

6.3.9.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.nrMmw.measurement.spath.repcap_stream_get()
driver.catalog.nrMmw.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:NRMMw:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:NRMMw:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.nrMmw.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.10 NrSub

class NrSubCls

NrSub 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.catalog.nrSub.clone()
```

Subgroups

6.3.10.1 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.catalog.nrSub.measurement.clone()
```

Subgroups

6.3.10.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.nrSub.measurement.spath.repcap_stream_get()
driver.catalog.nrSub.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:NRSub:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:NRSub:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.nrSub.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.11 System

SCPI Command :

```
CATalog:SYSTem:POSitioner
```

class SystemCls

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

get_positioner() → str

```
# SCPI: CATalog:SYSTem:POSitioner
value: str = driver.catalog.system.get_positioner()
```

No command help available

return
vendor_sn: No help available

Cloning the Group

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

Subgroups

6.3.11.1 Attenuation

class AttenuationCls

Attenuation 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.catalog.system.attenuation.clone()
```

Subgroups

6.3.11.1.1 CorrectionTable

SCPI Commands :

```
CATalog:SYSTem:ATTenuation:CTABle[:TENVironment]
CATalog:SYSTem:ATTenuation:CTABle:GLOBal
```

class CorrectionTableCls

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

get_globale() → str

```
# SCPI: CATalog:SYSTem:ATTenuation:CTABle:GLOBal
value: str = driver.catalog.system.attenuation.correctionTable.get_globale()
```

No command help available

return
name: No help available

get_tenvironment() → str

```
# SCPI: CATalog:SYSTem:ATTenuation:CTABle[:TENVironment]
value: str = driver.catalog.system.attenuation.correctionTable.get_
↪tenvironment()
```

No command help available

return
name: No help available

6.3.11.2 Reset

SCPI Command :

```
CATalog:SYSTem:RESet:PARTial
```

class ResetCls

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

get_partial() → List[str]

```
# SCPI: CATalog:SYSTem:RESet:PARTial
value: List[str] = driver.catalog.system.reset.get_partial()
```

No command help available

return
resetable_system_part: No help available

6.3.11.3 Rf42

class Rf42Cls

Rf42 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.catalog.system.rf42.clone()
```

Subgroups

6.3.11.3.1 Box

SCPI Command :

```
CATalog:SYSTem:RF42:BOX
```

class BoxCls

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

get(*serial_number: str*) → str

```
# SCPI: CATalog:SYSTem:RF42:BOX
value: str = driver.catalog.system.rf42.box.get(serial_number = 'abc')
```

No command help available

param serial_number
No help available

return
result: No help available

6.3.11.4 Rrhead

SCPI Command :

```
CATalog:SYSTem:RRHead
```

class RrheadCls

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

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

```
# SCPI: CATalog:SYSTem:RRHead
value: List[str] = driver.catalog.system.rrhead.get(filter_criteria = enums.
↪FilterCriteria.LOSupport)
```

No command help available

param filter_criteria
No help available

return
rrh_name: No help available

6.3.12 Tenvironment

SCPI Command :

```
CATalog:TENVironment:SPATh
```

class TenvironmentCls

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

get_spath() → List[str]

```
# SCPI: CATalog:TENVironment:SPATh
value: List[str] = driver.catalog.tenvironment.get_spath()
```

No command help available

return
name_signal_path: No help available

Cloning the Group

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

Subgroups

6.3.12.1 Connectors

SCPI Command :

```
CATalog:TENVironment:CONNectors
```

class ConnectorsCls

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

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

```
# SCPI: CATalog:TENVironment:CONNectors
value: List[str] = driver.catalog.tenvironment.connectors.get(name_style =␣
↪enums.NameStyle.FQName)
```

No command help available

param name_style
No help available

return
name_connector: No help available

6.3.13 Uwb

class UwbCls

Uwb 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.catalog.uwb.clone()
```

Subgroups

6.3.13.1 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.catalog.uwb.measurement.clone()
```

Subgroups

6.3.13.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.uwb.measurement.spath.repcap_stream_get()
driver.catalog.uwb.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:UWB:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:UWB:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.uwb.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.14 Wcdma

class WcdmaCls

Wcdma 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.catalog.wcdma.clone()
```

Subgroups

6.3.14.1 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.catalog.wcdma.measurement.clone()
```

Subgroups

6.3.14.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.wcdma.measurement.spath.repcap_stream_get()
driver.catalog.wcdma.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:WCDMa:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:WCDMa:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.wcdma.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.15 Wlan

class WlanCls

Wlan 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.catalog.wlan.clone()
```

Subgroups

6.3.15.1 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.catalog.wlan.measurement.clone()
```

Subgroups

6.3.15.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.wlan.measurement.spath.repcap_stream_get()
driver.catalog.wlan.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:WLAN:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:WLAN:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.wlan.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.3.16 Wpan

class WpanCls

Wpan 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.catalog.wpan.clone()
```

Subgroups

6.3.16.1 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.catalog.wpan.measurement.clone()
```

Subgroups

6.3.16.1.1 Spath<Stream>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.catalog.wpan.measurement.spath.repcap_stream_get()
driver.catalog.wpan.measurement.spath.repcap_stream_set(repcap.Stream.Nr1)
```

SCPI Command :

```
CATalog:WPAN:MEASurement<Instance>:SPATh<StreamNumber>
```

class SpathCls

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

get(*stream=Stream.Default*) → List[str]

```
# SCPI: CATalog:WPAN:MEASurement<Instance>:SPATh<StreamNumber>
value: List[str] = driver.catalog.wpan.measurement.spath.get(stream = repcap.
↪Stream.Default)
```

No command help available

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

return
name_signal_path: No help available

Cloning the Group

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

6.4 Configure

class ConfigureCls

Configure commands group definition. 208 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.1 Gprf

class GprfCls

Gprf commands group definition. 181 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.1.1 Generator

SCPI Command :

```
CONFigure:GPRF:GENerator<Instance>:TYPE
```

class GeneratorCls

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

get_type_py() → InstrumentType

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:TYPE
value: enums.InstrumentType = driver.configure.gprf.generator.get_type_py()
```

No command help available

return
instrument_type: No help available

set_type_py(*instrument_type: InstrumentType*) → None

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:TYPE
driver.configure.gprf.generator.set_type_py(instrument_type = enums.
↪InstrumentType.PROTocol)
```

No command help available

param instrument_type
No help available

Cloning the Group

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

Subgroups

6.4.1.1.1 Spath

SCPI Command :

```
CONFigure:GPRF:GENerator<Instance>:SPATh:BCSWitch
```

class SpathCls

Spath commands group definition. 4 total commands, 1 Subgroups, 1 group commands

get_bc_switch() → bool

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:BCSWitch
value: bool = driver.configure.gprf.generator.spath.get_bc_switch()
```

Enables signal broadcast to several connectors of an R&S CM-Z310A.

return
connect_switch: No help available

set_bc_switch(*connect_switch: bool*) → None

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:BCSWitch
driver.configure.gprf.generator.spath.set_bc_switch(connect_switch = False)
```

Enables signal broadcast to several connectors of an R&S CM-Z310A.

param connect_switch
No help available

Cloning the Group

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

Subgroups

6.4.1.1.1.1 Usage

SCPI Command :

```
CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe
```

class UsageCls

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

get_value() → List[bool]

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe
value: List[bool] = driver.configure.gprf.generator.spath.usage.get_value()
```

Activates or deactivates the individual RF connectors of the active connector group (global definition) . Select the connector group via method RsCMPX_Gprf.Route.Gprf.Generator.Spath.value. Query a list of the connectors of the connector group via method RsCMPX_Gprf.Catalog.Gprf.Generator.Spath.Group.connector.

return
enable: Comma-separated list of values, one value per connector of the connector group. ON: activate the connector OFF: deactivate the connector

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

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe
driver.configure.gprf.generator.spath.usage.set_value(enable = [True, False,␣
↪True])
```

Activates or deactivates the individual RF connectors of the active connector group (global definition) . Select the connector group via method RsCMPX_Gprf.Route.Gprf.Generator.Spath.value. Query a list of the connectors of the connector group via method RsCMPX_Gprf.Catalog.Gprf.Generator.Spath.Group.connector.

param enable
Comma-separated list of values, one value per connector of the connector group. ON: activate the connector OFF: deactivate the connector

Cloning the Group

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

Subgroups

6.4.1.1.1.2 Bench<Bench>

RepCap Settings

```
# Range: Nr1 .. Nr20
rc = driver.configure.gprf.generator.spath.usage.bench.repcap_bench_get()
driver.configure.gprf.generator.spath.usage.bench.repcap_bench_set(repcap.Bench.Nr1)
```

SCPI Command :

```
CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe:BENCh<nr>
```

class BenchCls

Bench commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: Bench, default value after init: Bench.Nr1

get(*bench=Bench.Default*) → List[bool]

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe:BENCh<nr>
value: List[bool] = driver.configure.gprf.generator.spath.usage.bench.get(bench␣
↪= repcap.Bench.Default)
```

Activates or deactivates the individual RF connectors of the connector group <no> (global definition) .

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

return
enable: Comma-separated list of values, one value per connector of the connector group. ON: activate the connector OFF: deactivate the connector

set(*enable: List[bool]*, *bench=Bench.Default*) → None

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe:BENCh<nr>
driver.configure.gprf.generator.spath.usage.bench.set(enable = [True, False,␣
↪True], bench = repcap.Bench.Default)
```

Activates or deactivates the individual RF connectors of the connector group <no> (global definition) .

param enable
Comma-separated list of values, one value per connector of the connector group. ON: activate the connector OFF: deactivate the connector

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

Cloning the Group

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

Subgroups

6.4.1.1.1.3 Tx

class TxCls
Tx 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.gprf.generator.spath.usage.bench.tx.clone()
```

Subgroups

6.4.1.1.1.4 Single

SCPI Command :

```
CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe:BENCh<nr>:TX:SINGle
```

class SingleCls
Single commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*tx_index: float*, *bench=Bench.Default*) → bool

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe:BENCh<nr>:TX:SINGle
value: bool = driver.configure.gprf.generator.spath.usage.bench.tx.single.
↪get(tx_index = 1.0, bench = repcap.Bench.Default)
```

Activates or deactivates the RF connector RF<no>.<TxIndex>+1. Example: <no>=2 plus <TxIndex>=4 means connector RF2.5.

param tx_index
No help available

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

return
usage: ON: activate the connector OFF: deactivate the connector

set(*tx_index: float*, *usage: bool*, *bench=Bench.Default*) → None

```
# SCPI: CONFigure:GPRF:GENerator<Instance>:SPATh:USAGe:BENCh<nr>:TX:SINGle
driver.configure.gprf.generator.spath.usage.bench.tx.single.set(tx_index = 1.0,␣
↪usage = False, bench = repcap.Bench.Default)
```

Activates or deactivates the RF connector RF<no>.<TxIndex>+1. Example: <no>=2 plus <TxIndex>=4 means connector RF2.5.

param tx_index
No help available

param usage
ON: activate the connector OFF: deactivate the connector

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

6.4.1.2 Measurement

class MeasurementCls

Measurement commands group definition. 176 total commands, 12 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.1.2.1 Canalyzer

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:CANalyzer:MNAMe
CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SEGMent
CONFigure:GPRF:MEASurement<Instance>:CANalyzer:STEP
```

class CanalyzerCls

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

get_mname() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:MNAMe
value: str = driver.configure.gprf.measurement.canalyzer.get_mname()
```

Queries which firmware application has captured the data.

return
meas_name: No help available

get_segment() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SEGMent
value: int = driver.configure.gprf.measurement.canalyzer.get_segment()
```

Selects a segment of a list mode measurement for result analysis. The selection affects the GUI contents and the contents stored by the command method RsCMPX_Gprf.Configure.Gprf.Measurement.Canalyzer.IqFile.set.

return
segment: Segment number

get_step() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:STEP
value: int = driver.configure.gprf.measurement.canalyzer.get_step()
```

Selects a step of a list mode measurement for result analysis. The selection affects the GUI contents and the contents stored by the command method RsCMPX_Gprf.Configure.Gprf.Measurement.Canalyzer.IqFile.set.

return
step: Step number

set_segment(*segment: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SEGMent
driver.configure.gprf.measurement.canalyzer.set_segment(segment = 1)
```

Selects a segment of a list mode measurement for result analysis. The selection affects the GUI contents and the contents stored by the command method RsCMPX_Gprf.Configure.Gprf.Measurement.Canalyzer.IqFile.set.

param segment
Segment number

set_step(*step: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:STEP
driver.configure.gprf.measurement.canalyzer.set_step(step = 1)
```

Selects a step of a list mode measurement for result analysis. The selection affects the GUI contents and the contents stored by the command method RsCMPX_Gprf.Configure.Gprf.Measurement.Canalyzer.IqFile.set.

param step
Step number

Cloning the Group

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

Subgroups

6.4.1.2.1.1 IqFile

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:CANalyzer:IQFile
```

class IqFileCls

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

get() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:IQFile
value: str = driver.configure.gprf.measurement.canalyzer.iqFile.get()
```

Saves the I/Q data for the current step to the selected file.

return
filename_return: No help available

set(*filename: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:IQFile
driver.configure.gprf.measurement.canalyzer.iqFile.set(filename = 'abc')
```

Saves the I/Q data for the current step to the selected file.

param filename
Name and path of the target file.

6.4.1.2.1.2 Sall

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SALL:IQFolder
CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SALL:WTFolder
```

class SallCls

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

get_iq_folder() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SALL:IQFolder
value: str = driver.configure.gprf.measurement.canalyzer.sall.get_iq_folder()
```

Selects a folder for storage of all buffer contents to files (I/Q data for all segments) . The default folder is @USERDATA/captureanalyzer. You can only select a subfolder of this default folder. All files within the selected folder are deleted when the capture buffer analyzer writes the result files.

return
folder_name: Name and path of the folder.

get_wt_folder() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SALL:WTFolder
value: bool = driver.configure.gprf.measurement.canalyzer.sall.get_wt_folder()
```

Enables or disables saving of all buffer contents to files (I/Q data for all segments) . With <WriteToFolder> = ON, the files are stored when the capture buffer analyzer is started.

return
write_to_folder: No help available

set_iq_folder(*folder_name: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SALL:IQFolder
driver.configure.gprf.measurement.canalyzer.sall.set_iq_folder(folder_name =
↪'abc')
```

Selects a folder for storage of all buffer contents to files (I/Q data for all segments) . The default folder is @USERDATA/captureanalyzer. You can only select a subfolder of this default folder. All files within the selected folder are deleted when the capture buffer analyzer writes the result files.

param folder_name
Name and path of the folder.

set_wt_folder(*write_to_folder: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CANalyzer:SALL:WTFolder
driver.configure.gprf.measurement.canalyzer.sall.set_wt_folder(write_to_folder␣
↪= False)
```

Enables or disables saving of all buffer contents to files (I/Q data for all segments) . With <WriteToFolder> = ON, the files are stored when the capture buffer analyzer is started.

param write_to_folder
No help available

6.4.1.2.2 Correction

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:CORR
```

class CorrectionCls

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

get() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CORR
value: str = driver.configure.gprf.measurement.correction.get()
```

No command help available

return
op_reply_code: No help available

set(*rx_corr_string: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CORR
driver.configure.gprf.measurement.correction.set(rx_corr_string = 'abc')
```

No command help available

param rx_corr_string
No help available

6.4.1.2.3 ExtPwrSensor

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:EPSensor:TOUT
CONFigure:GPRF:MEASurement<Instance>:EPSensor:RESolution
CONFigure:GPRF:MEASurement<Instance>:EPSensor:SCOunt
CONFigure:GPRF:MEASurement<Instance>:EPSensor:REPetition
CONFigure:GPRF:MEASurement<Instance>:EPSensor:FREQuency
```

class ExtPwrSensorCls

ExtPwrSensor commands group definition. 12 total commands, 3 Subgroups, 5 group commands

get_frequency() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:FREQuency
value: float = driver.configure.gprf.measurement.extPwrSensor.get_frequency()
```

Specifies the input frequency at the power sensor.

return
correction_freq: No help available

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:REPetition
value: enums.Repeat = driver.configure.gprf.measurement.extPwrSensor.get_
↪repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

return
repetition: SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_resolution() → PwrSensorResolution

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RESolution
value: enums.PwrSensorResolution = driver.configure.gprf.measurement.
↪extPwrSensor.get_resolution()
```

Defines the number of digits of the displayed power results. This command does not affect the remote control results.

return
resolution: PD0: 1 (results rounded to 1 dB) PD1: 0.1 PD2: 0.01 PD3: 0.001

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:SCOunt
value: int = driver.configure.gprf.measurement.extPwrSensor.get_scount()
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

return
statistic_count: No help available

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:TOUT
value: float = driver.configure.gprf.measurement.extPwrSensor.get_timeout()
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
tcd_timeout: No help available

set_frequency(*correction_freq: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:FREQuency
driver.configure.gprf.measurement.extPwrSensor.set_frequency(correction_freq =␣
↪1.0)
```

Specifies the input frequency at the power sensor.

param correction_freq
No help available

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:REPetition
driver.configure.gprf.measurement.extPwrSensor.set_repetition(repetition =␣
↪enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

param repetition
SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_resolution(*resolution: PwrSensorResolution*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RESolution
driver.configure.gprf.measurement.extPwrSensor.set_resolution(resolution =␣
↪enums.PwrSensorResolution.PD0)
```

Defines the number of digits of the displayed power results. This command does not affect the remote control results.

param resolution
PD0: 1 (results rounded to 1 dB) PD1: 0.1 PD2: 0.01 PD3: 0.001

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:SCOunt
driver.configure.gprf.measurement.extPwrSensor.set_scount(statistic_count = 1)
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

param statistic_count
No help available

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:TOUT
driver.configure.gprf.measurement.extPwrSensor.set_timeout(tcd_timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param tcd_timeout
No help available

Cloning the Group

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

Subgroups

6.4.1.2.3.1 Attenuation

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation:STATe
CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation
```

class AttenuationCls

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

get_state() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation:STATe
value: bool = driver.configure.gprf.measurement.extPwrSensor.attenuation.get_
↪state()
```

Enables or disables the result correction for an external input attenuation.

return
attenuator_state: No help available

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation
value: float = driver.configure.gprf.measurement.extPwrSensor.attenuation.get_
↪value()
```

Specifies an external input attenuation factor for correction of the power results.

return
attenuation: No help available

set_state(*attenuator_state: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation:STATe
driver.configure.gprf.measurement.extPwrSensor.attenuation.set_state(attenuator_
↪state = False)
```

Enables or disables the result correction for an external input attenuation.

param attenuator_state
No help available

set_value(*attenuation: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation
driver.configure.gprf.measurement.extPwrSensor.attenuation.set_
↪value(attenuation = 1.0)
```

Specifies an external input attenuation factor for correction of the power results.

param attenuation
No help available

6.4.1.2.3.2 Auto

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:EPSensor:AUTO:MTIMe
CONFigure:GPRF:MEASurement<Instance>:EPSensor:AUTO:NSR
```

class AutoCls

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

get_mtime() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AUTO:MTIMe
value: float = driver.configure.gprf.measurement.extPwrSensor.auto.get_mtime()
```

No command help available

return
meas_time: No help available

get_nsr() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AUTO:NSR
value: float = driver.configure.gprf.measurement.extPwrSensor.auto.get_nsr()
```

No command help available

return
nsr: No help available

set_mtime(*meas_time: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AUTO:MTIMe
driver.configure.gprf.measurement.extPwrSensor.auto.set_mtime(meas_time = 1.0)
```

No command help available

param meas_time
No help available

set_nsr(*nsr: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AUTO:NSR
driver.configure.gprf.measurement.extPwrSensor.auto.set_nsr(nsr = 1.0)
```

No command help available

param nsr
No help available

6.4.1.2.3.3 Average

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:MODE
CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:COUNt
CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:APERture
```

class AverageCls

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

get_aperture() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:APERture
value: float = driver.configure.gprf.measurement.extPwrSensor.average.get_
↪aperture()
```

No command help available

return
aperture: No help available

get_count() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:COUNt
value: int = driver.configure.gprf.measurement.extPwrSensor.average.get_count()
```

No command help available

return
average_count: No help available

get_mode() → ExtPwrSensorAvgMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:MODE
value: enums.ExtPwrSensorAvgMode = driver.configure.gprf.measurement.
↪extPwrSensor.average.get_mode()
```

No command help available

return
mode: No help available

set_aperture(*aperture: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:APERture
driver.configure.gprf.measurement.extPwrSensor.average.set_aperture(aperture =␣
↪1.0)
```

No command help available

param aperture
No help available

set_count(*average_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:COUNt
driver.configure.gprf.measurement.extPwrSensor.average.set_count(average_count␣
↪= 1)
```

No command help available

param average_count
No help available

set_mode(*mode: ExtPwrSensorAvgMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:MODE
driver.configure.gprf.measurement.extPwrSensor.average.set_mode(mode = enums.
↪ExtPwrSensorAvgMode.MANual)
```

No command help available

param mode
No help available

6.4.1.2.4 FftSpecAn

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:AMODe
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:DETector
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FFTLength
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FSPan
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MOEXception
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:REPetition
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
```

class FftSpecAnCls

FftSpecAn commands group definition. 10 total commands, 1 Subgroups, 8 group commands

get_amode() → AveragingMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:AMODe
value: enums.AveragingMode = driver.configure.gprf.measurement.fftSpecAn.get_
↪amode()
```

Selects the averaging mode for the average spectrum trace.

return
averaging_mode: LINear: averaging of linear power values LOGarithmic: averaging of logarithmic power values

get_detector() → DetectorBasic

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:DETector
value: enums.DetectorBasic = driver.configure.gprf.measurement.fftSpecAn.get_
↪detector()
```

Defines how the spectrum diagram is calculated from the frequency domain samples.

return
detector: PEAK: The peak value of adjacent samples is used. RMS: The RMS value of adjacent samples is used.

get_fft_length() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FFTLength
value: int = driver.configure.gprf.measurement.fftSpecAn.get_fft_length()
```

Selects the number of samples recorded per measurement interval.

return
length: Only the following values can be configured: 1024, 2048, 4096, 8192, 16384 Other values are rounded to the next allowed value.

get_fspan() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FSPan
value: float = driver.configure.gprf.measurement.fftSpecAn.get_fspan()
```

Configures the frequency span of the FFT spectrum analyzer.

return
frequency_span: Only the following values can be configured, all values in MHz: IF unit: 10, 20, 40, 80, 160, 250, 500, 1000 RF unit: 10, 20, 40, 80, 160, 250 R&S CMW: 1.25, 2.5, 5, 10, 20, 40, 80, 160 Other values are rounded to the next allowed value.

get_mo_exception() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MOEXception
value: bool = driver.configure.gprf.measurement.fftSpecAn.get_mo_exception()
```

Specifies whether measurement results that the CMX500 identifies as faulty or inaccurate are rejected.

return
meas_on_exception: OFF: Faulty results are rejected. ON: Results are never rejected.

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:REPetition
value: enums.Repeat = driver.configure.gprf.measurement.fftSpecAn.get_
↪repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

return
repetition: SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
value: int = driver.configure.gprf.measurement.fftSpecAn.get_scount()
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

return
statistic_count: Number of measurement intervals

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
value: float = driver.configure.gprf.measurement.fftSpecAn.get_timeout()
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
tcd_timeout: No help available

set_amode(*averaging_mode: AveragingMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:AMODe
driver.configure.gprf.measurement.fftSpecAn.set_amode(averaging_mode = enums.
↪AveragingMode.LINear)
```

Selects the averaging mode for the average spectrum trace.

param averaging_mode
LINear: averaging of linear power values LOGarithmic: averaging of logarithmic power values

set_detector(*detector: DetectorBasic*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:DETector
driver.configure.gprf.measurement.fftSpecAn.set_detector(detector = enums.
↪DetectorBasic.PEAK)
```

Defines how the spectrum diagram is calculated from the frequency domain samples.

param detector
PEAK: The peak value of adjacent samples is used. RMS: The RMS value of adjacent samples is used.

set_fft_length(*length: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FFTLength
driver.configure.gprf.measurement.fftSpecAn.set_fft_length(length = 1)
```

Selects the number of samples recorded per measurement interval.

param length
Only the following values can be configured: 1024, 2048, 4096, 8192, 16384 Other values are rounded to the next allowed value.

set_fspan(*frequency_span: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FSPan
driver.configure.gprf.measurement.fftSpecAn.set_fspan(frequency_span = 1.0)
```

Configures the frequency span of the FFT spectrum analyzer.

param frequency_span
Only the following values can be configured, all values in MHz: IF unit: 10, 20, 40, 80, 160, 250, 500, 1000 RF unit: 10, 20, 40, 80, 160, 250 R&S CMW: 1.25, 2.5, 5, 10, 20, 40, 80, 160 Other values are rounded to the next allowed value.

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MOEXception
driver.configure.gprf.measurement.fftSpecAn.set_mo_exception(meas_on_exception␣
↪= False)
```

Specifies whether measurement results that the CMX500 identifies as faulty or inaccurate are rejected.

param meas_on_exception
OFF: Faulty results are rejected. ON: Results are never rejected.

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:REPetition
driver.configure.gprf.measurement.fftSpecAn.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

param repetition
SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
driver.configure.gprf.measurement.fftSpecAn.set_scount(statistic_count = 1)
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

param statistic_count
Number of measurement intervals

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
driver.configure.gprf.measurement.fftSpecAn.set_timeout(tcd_timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has

completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param tcd_timeout
No help available

Cloning the Group

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

Subgroups

6.4.1.2.4.1 PeakSearch

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch:NOAMarkers
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch
```

class PeakSearchCls

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

class ValueStruct

Structure for setting input parameters. Fields:

- Full_Span_Enable_0: bool: Enable full-span search for marker 0. OFF: Search the configured range. ON: Search the full span and ignore the configured range.
- Peak_Range_From_0: float: Lower end of the search range for marker 0.
- Peak_Range_To_0: float: Upper end of the search range for marker 0.
- Full_Span_Enable_1: bool: Enable full-span search for marker 1.
- Peak_Range_From_1: float: Lower end of the search range for marker 1.
- Peak_Range_To_1: float: Upper end of the search range for marker 1.
- Full_Span_Enable_2: bool: Enable full-span search for marker 2.
- Peak_Range_From_2: float: Lower end of the search range for marker 2.
- Peak_Range_To_2: float: Upper end of the search range for marker 2.
- Full_Span_Enable_3: bool: Enable full-span search for marker 3.
- Peak_Range_From_3: float: Lower end of the search range for marker 3.
- Peak_Range_To_3: float: Upper end of the search range for marker 3.
- Full_Span_Enable_4: bool: Enable full-span search for marker 4.
- Peak_Range_From_4: float: Lower end of the search range for marker 4.

- Peak_Range_To_4: float: Upper end of the search range for marker 4.

get_noa_markers() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch:NOAMarkers
value: int = driver.configure.gprf.measurement.fftSpecAn.peakSearch.get_noa_
↪markers()
```

Defines the number of active markers for the peak search.

return
no_active_markers: No help available

get_value() → ValueStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch
value: ValueStruct = driver.configure.gprf.measurement.fftSpecAn.peakSearch.get_
↪value()
```

Defines the peak search ranges. The maximum allowed search ranges depend on the frequency span: –span/2 to span/2.

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

set_noa_markers(*no_active_markers: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch:NOAMarkers
driver.configure.gprf.measurement.fftSpecAn.peakSearch.set_noa_markers(no_
↪active_markers = 1)
```

Defines the number of active markers for the peak search.

param no_active_markers
No help available

set_value(*value: ValueStruct*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch
structure = driver.configure.gprf.measurement.fftSpecAn.peakSearch.ValueStruct()
structure.Full_Span_Enable_0: bool = False
structure.Peak_Range_From_0: float = 1.0
structure.Peak_Range_To_0: float = 1.0
structure.Full_Span_Enable_1: bool = False
structure.Peak_Range_From_1: float = 1.0
structure.Peak_Range_To_1: float = 1.0
structure.Full_Span_Enable_2: bool = False
structure.Peak_Range_From_2: float = 1.0
structure.Peak_Range_To_2: float = 1.0
structure.Full_Span_Enable_3: bool = False
structure.Peak_Range_From_3: float = 1.0
structure.Peak_Range_To_3: float = 1.0
structure.Full_Span_Enable_4: bool = False
structure.Peak_Range_From_4: float = 1.0
structure.Peak_Range_To_4: float = 1.0
driver.configure.gprf.measurement.fftSpecAn.peakSearch.set_value(value =␣
↪structure)
```

Defines the peak search ranges. The maximum allowed search ranges depend on the frequency span: –span/2 to span/2.

param value
see the help for ValueStruct structure arguments.

6.4.1.2.5 IqRecorder

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:SRATe
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MODE
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:RATio
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:BYPass
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FORMat
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MUNit
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:USER
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQFile
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:WTFile
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:INIFile
```

class IqRecorderCls

IqRecorder commands group definition. 27 total commands, 5 Subgroups, 11 group commands

get_bypass() → IqRecBypass

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:BYPass
value: enums.IqRecBypass = driver.configure.gprf.measurement.iqRecorder.get_
↪bypass()
```

No command help available

return
bypass: No help available

get_format_py() → IqFormat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FORMat
value: enums.IqFormat = driver.configure.gprf.measurement.iqRecorder.get_format_
↪py()
```

Selects the coordinate system for the I/Q recorder results.

return
format_py: IQ: Cartesian coordinates (I- and Q-axis) RPHI: polar coordinates (radius R and angle PHI)

get_ini_file() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:INIFile
value: str = driver.configure.gprf.measurement.iqRecorder.get_ini_file()
```

No command help available

return
ini_file: No help available

get_iq_file() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQFile
value: str = driver.configure.gprf.measurement.iqRecorder.get_iq_file()
```

Selects a file for storage of the I/Q recorder results in binary format.

return
iq_save_file: Name and path of the file. The extension *.iqw is appended automatically.

get_mode() → MeasurementMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MODE
value: enums.MeasurementMode = driver.configure.gprf.measurement.iqRecorder.get_
↪mode()
```

No command help available

return
measurement_mode: No help available

get_munit() → MagnitudeUnit

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MUNit
value: enums.MagnitudeUnit = driver.configure.gprf.measurement.iqRecorder.get_
↪munit()
```

Selects the magnitude unit for the measurement results.

return
magnitude_unit: Voltage units or raw I/Q data relative to full-scale.

get_ratio() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:RATio
value: float = driver.configure.gprf.measurement.iqRecorder.get_ratio()
```

Specifies a factor to reduce the sampling rate and to increase the measurement duration. The sampling rate resulting from the filter settings is multiplied with the specified ratio.

return
ratio: No help available

get_symbol_rate() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:SRATe
value: float = driver.configure.gprf.measurement.iqRecorder.get_symbol_rate()
```

No command help available

return
sample_rate: No help available

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
value: float = driver.configure.gprf.measurement.iqRecorder.get_timeout()
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
tcd_timeout: No help available

get_user() → UserDebugMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:USER
value: enums.UserDebugMode = driver.configure.gprf.measurement.iqRecorder.get_
↪user()
```

No command help available

return
user_mode: No help available

get_wt_file() → FileSave

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:WTFile
value: enums.FileSave = driver.configure.gprf.measurement.iqRecorder.get_wt_
↪file()
```

Selects whether the results are written to a file, to the memory or both. For file selection, see method RsCMPX_Gprf. Configure.Gprf.Measurement.IqRecorder.iqFile.

return
write_to_iq_file: OFF: The results are only stored in the memory. ON: The results are stored in the memory and in a file. ONLY: The results are only stored in a file.

set_bypass(*bypass: IqRecBypass*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:BYPass
driver.configure.gprf.measurement.iqRecorder.set_bypass(bypass = enums.
↪IqRecBypass.BIT)
```

No command help available

param bypass
No help available

set_format_py(*format_py: IqFormat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FORMat
driver.configure.gprf.measurement.iqRecorder.set_format_py(format_py = enums.
↪IqFormat.IQ)
```

Selects the coordinate system for the I/Q recorder results.

param format_py
IQ: Cartesian coordinates (I- and Q-axis) RPHI: polar coordinates (radius R and angle PHI)

set_ini_file(*ini_file: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:INIFile
driver.configure.gprf.measurement.iqRecorder.set_ini_file(ini_file = 'abc')
```

No command help available

param ini_file
No help available

set_iq_file(*iq_save_file: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQFile
driver.configure.gprf.measurement.iqRecorder.set_iq_file(iq_save_file = 'abc')
```

Selects a file for storage of the I/Q recorder results in binary format.

param iq_save_file
Name and path of the file. The extension *.iqw is appended automatically.

set_mode(*measurement_mode: MeasurementMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MODE
driver.configure.gprf.measurement.iqRecorder.set_mode(measurement_mode = enums.
↪MeasurementMode.NORMal)
```

No command help available

param measurement_mode
No help available

set_munit(*magnitude_unit: MagnitudeUnit*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MUNit
driver.configure.gprf.measurement.iqRecorder.set_munit(magnitude_unit = enums.
↪MagnitudeUnit.RAW)
```

Selects the magnitude unit for the measurement results.

param magnitude_unit
Voltage units or raw I/Q data relative to full-scale.

set_ratio(*ratio: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:RATio
driver.configure.gprf.measurement.iqRecorder.set_ratio(ratio = 1.0)
```

Specifies a factor to reduce the sampling rate and to increase the measurement duration. The sampling rate resulting from the filter settings is multiplied with the specified ratio.

param ratio
No help available

set_symbol_rate(*sample_rate: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:SRATe
driver.configure.gprf.measurement.iqRecorder.set_symbol_rate(sample_rate = 1.0)
```

No command help available

param sample_rate
No help available

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
driver.configure.gprf.measurement.iqRecorder.set_timeout(tcd_timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param tcd_timeout
No help available

set_user(*user_mode: UserDebugMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:USER
driver.configure.gprf.measurement.iqRecorder.set_user(user_mode = enums.
↪UserDebugMode.DEBug)
```

No command help available

param user_mode
No help available

set_wt_file(*write_to_iq_file: FileSave*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:WTFile
driver.configure.gprf.measurement.iqRecorder.set_wt_file(write_to_iq_file =␣
↪enums.FileSave.OFF)
```

Selects whether the results are written to a file, to the memory or both. For file selection, see method RsCMPX_Gprf. Configure.Gprf.Measurement.IqRecorder.iqFile.

param write_to_iq_file
OFF: The results are only stored in the memory. ON: The results are stored in the memory and in a file. ONLY: The results are only stored in a file.

Cloning the Group

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

Subgroups

6.4.1.2.5.1 Capture

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:CAPTure
```

class CaptureCls

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

class CaptureStruct

Response structure. Fields:

- Capt_Samp_Bef_Trig: int: Samples before trigger event
- Capt_Samp_Aft_Trig: int: Samples after trigger event

get() → CaptureStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:CAPTure
value: CaptureStruct = driver.configure.gprf.measurement.iqRecorder.capture.
↪get()
```

Selects the number of samples to be recorded before and after the trigger event. Configure the two settings so that their sum does not exceed the maximum number of samples.

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

set(*capt_samp_bef_trig: int*, *capt_samp_aft_trig: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:CAPTure
driver.configure.gprf.measurement.iqRecorder.capture.set(capt_samp_bef_trig = 1,
↪ capt_samp_aft_trig = 1)
```

Selects the number of samples to be recorded before and after the trigger event. Configure the two settings so that their sum does not exceed the maximum number of samples.

param capt_samp_bef_trig
Samples before trigger event

param capt_samp_aft_trig
Samples after trigger event

6.4.1.2.5.2 FilterPy

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:TYPE
```

class FilterPyCls

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

get_type_py() → RbwFilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:TYPE
value: enums.RbwFilterType = driver.configure.gprf.measurement.iqRecorder.
↪filterPy.get_type_py()
```

Selects the IF filter type.

return
filter_type: BANDpass: bandpass filter GAUSs: filter of Gaussian shape

set_type_py(*filter_type: RbwFilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:TYPE
driver.configure.gprf.measurement.iqRecorder.filterPy.set_type_py(filter_type =␣
↪enums.RbwFilterType.BANDpass)
```

Selects the IF filter type.

param filter_type
BANDpass: bandpass filter GAUSs: filter of Gaussian shape

Cloning the Group

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

Subgroups

6.4.1.2.5.3 Bandpass

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:BANDpass:BWIDth
```

class BandpassCls

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

get_bandwidth() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:BANDpass:BWIDth
value: float = driver.configure.gprf.measurement.iqRecorder.filterPy.bandpass.
↪get_bandwidth()
```

Selects the bandwidth for a bandpass filter.

return
bandpass_bw: Only the following values can be configured: IF unit: 7.8125, 15.625, 31.25, 62.5, 125, 250, 500, 1000 MHz RF unit: 7.8125, 15.625, 31.25, 62.5, 125, 250 MHz R&S CMW: 1, 10, 100 kHz; 1, 10, 40, 160 MHz Other values are rounded to the next allowed value.

set_bandwidth(*bandpass_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:BANDpass:BWIDth
driver.configure.gprf.measurement.iqRecorder.filterPy.bandpass.set_
↪bandwidth(bandpass_bw = 1.0)
```

Selects the bandwidth for a bandpass filter.

param bandpass_bw
Only the following values can be configured: IF unit: 7.8125, 15.625, 31.25, 62.5, 125, 250, 500, 1000 MHz RF unit: 7.8125, 15.625, 31.25, 62.5, 125, 250 MHz R&S CMW: 1, 10, 100 kHz; 1, 10, 40, 160 MHz Other values are rounded to the next allowed value.

6.4.1.2.5.4 Gauss

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:GAUSs:BWIDth
```

class GaussCls

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

get_bandwidth() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:GAUSs:BWIDth
value: float = driver.configure.gprf.measurement.iqRecorder.filterPy.gauss.get_
↪bandwidth()
```

Selects the bandwidth for a filter of Gaussian shape.

return
gauss_bw: Only the following values can be configured: 1 kHz, 10 kHz, 100 kHz, 1 MHz, 10 MHz Other values are rounded to the next allowed value.

set_bandwidth(*gauss_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:GAUSs:BWIDth
driver.configure.gprf.measurement.iqRecorder.filterPy.gauss.set_bandwidth(gauss_
↪bw = 1.0)
```

Selects the bandwidth for a filter of Gaussian shape.

param gauss_bw
Only the following values can be configured: 1 kHz, 10 kHz, 100 kHz, 1 MHz, 10 MHz Other values are rounded to the next allowed value.

6.4.1.2.5.5 IqSettings

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQSettings:SRATe
```

class IqSettingsCls

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

get_symbol_rate() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQSettings:SRATe
value: float = driver.configure.gprf.measurement.iqRecorder.iqSettings.get_
↪symbol_rate()
```

No command help available

return
sample_rate: No help available

set_symbol_rate(*sample_rate: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQSettings:SRATe
driver.configure.gprf.measurement.iqRecorder.iqSettings.set_symbol_rate(sample_
↪rate = 1.0)
```

No command help available

param sample_rate
No help available

6.4.1.2.5.6 ListPy

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:SLENgth
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:COUNt
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:STARt
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:STOP
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST
```

class ListPyCls

ListPy commands group definition. 10 total commands, 3 Subgroups, 5 group commands

get_count() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:COUNt
value: int = driver.configure.gprf.measurement.iqRecorder.listPy.get_count()
```

No command help available

return
result_count: No help available

get_slength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:SLENgth
value: float = driver.configure.gprf.measurement.iqRecorder.listPy.get_slength()
```

No command help available

return
step_length: No help available

get_start() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:STARt
value: int = driver.configure.gprf.measurement.iqRecorder.listPy.get_start()
```

No command help available

return
start_index: No help available

get_stop() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:STOP
value: int = driver.configure.gprf.measurement.iqRecorder.listPy.get_stop()
```

No command help available

return
stop_index: No help available

get_value() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST
value: bool = driver.configure.gprf.measurement.iqRecorder.listPy.get_value()
```

No command help available

return
enable_list_mode: No help available

set_slength(*step_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:SLENgth
driver.configure.gprf.measurement.iqRecorder.listPy.set_slength(step_length = 1.
↪0)
```

No command help available

param step_length
No help available

set_start(*start_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:STARt
driver.configure.gprf.measurement.iqRecorder.listPy.set_start(start_index = 1)
```

No command help available

param start_index
No help available

set_stop(*stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:STOP
driver.configure.gprf.measurement.iqRecorder.listPy.set_stop(stop_index = 1)
```

No command help available

param stop_index
No help available

set_value(*enable_list_mode: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST
driver.configure.gprf.measurement.iqRecorder.listPy.set_value(enable_list_mode␣
↪= False)
```

No command help available

param enable_list_mode
No help available

Cloning the Group

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

Subgroups

6.4.1.2.5.7 EnvelopePower

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:ENPower
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:ENPower:ALL
```

class EnvelopePowerCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:ENPower
value: float = driver.configure.gprf.measurement.iqRecorder.listPy.
↪envelopePower.get(index = 1)
```

No command help available

param index
No help available

return
exp_nom_power: No help available

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:ENPower:ALL
value: List[float] = driver.configure.gprf.measurement.iqRecorder.listPy.
↪envelopePower.get_all()
```

No command help available

return
exp_nom_power: No help available

set(*index: int*, *exp_nom_power: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:ENPower
driver.configure.gprf.measurement.iqRecorder.listPy.envelopePower.set(index = 1,
↪ exp_nom_power = 1.0)
```

No command help available

param index
No help available

param exp_nom_power
No help available

set_all(*exp_nom_power: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:ENPower:ALL
driver.configure.gprf.measurement.iqRecorder.listPy.envelopePower.set_all(exp_
↪nom_power = [1.1, 2.2, 3.3])
```

No command help available

param exp_nom_power
No help available

6.4.1.2.5.8 Frequency

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:FREQuency
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:FREQuency:ALL
```

class FrequencyCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:FREQuency
value: float = driver.configure.gprf.measurement.iqRecorder.listPy.frequency.
↪get(index = 1)
```

No command help available

param index
No help available

return
frequency: No help available

`get_all`() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:FREQuency:ALL
value: List[float] = driver.configure.gprf.measurement.iqRecorder.listPy.
↪frequency.get_all()
```

No command help available

return
frequency: No help available

`set`(*index: int*, *frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:FREQuency
driver.configure.gprf.measurement.iqRecorder.listPy.frequency.set(index = 1,␣
↪frequency = 1.0)
```

No command help available

param index
No help available

param frequency
No help available

`set_all`(*frequency: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:FREQuency:ALL
driver.configure.gprf.measurement.iqRecorder.listPy.frequency.set_all(frequency␣
↪= [1.1, 2.2, 3.3])
```

No command help available

param frequency
No help available

6.4.1.2.5.9 Sstop

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:SSTop
```

`class SstopCls`

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

`class SstopStruct`

Response structure. Fields:

- Start_Index: int: No parameter help available
- Stop_Index: int: No parameter help available

get() → SstopStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:SSTop
value: SstopStruct = driver.configure.gprf.measurement.iqRecorder.listPy.sstop.
↪get()
```

No command help available

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

set(*start_index: int*, *stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:LIST:SSTop
driver.configure.gprf.measurement.iqRecorder.listPy.sstop.set(start_index = 1,␣
↪stop_index = 1)
```

No command help available

param start_index
No help available

param stop_index
No help available

6.4.1.2.5.10 Trigger

SCPI Command :

```
[CONFigure]:GPRF:MEASurement<Instance>:IQRecorder:TRIGger:SOURce
```

class TriggerCls

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

get_source() → TriggerSource

```
# SCPI: [CONFigure]:GPRF:MEASurement<Instance>:IQRecorder:TRIGger:SOURce
value: enums.TriggerSource = driver.configure.gprf.measurement.iqRecorder.
↪trigger.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: TriggerSource*) → None

```
# SCPI: [CONFigure]:GPRF:MEASurement<Instance>:IQRecorder:TRIGger:SOURce
driver.configure.gprf.measurement.iqRecorder.trigger.set_source(trigger = enums.
↪TriggerSource.EXTernal)
```

No command help available

param trigger
No help available

6.4.1.2.6 IqVsSlot

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:TOUT
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:REPetition
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SCOunt
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:MLENgth
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SLENgth
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:FTYPe
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:FELimit
```

class IqVsSlotCls

IqVsSlot commands group definition. 19 total commands, 2 Subgroups, 7 group commands

get_fe_limit() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:FELimit
value: float = driver.configure.gprf.measurement.iqVsSlot.get_fe_limit()
```

Defines the frequency estimation limit as a signal level relative to the expected nominal power. Steps with a level below this limit are not used for the frequency correction and do not contribute to the frequency results.

return
limit: Range-100 dB to 0 dB*RST-100 dBDefault unitdB

get_ftype() → FilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:FTYPe
value: enums.FilterType = driver.configure.gprf.measurement.iqVsSlot.get_ftype()
```

Selects the IF filter type.

return
filter_type: IF unit: B1MHz | B10Mhz RF unit: B1MHz | B10Mhz R&S CMW: GAUSs | NYQuist | NY1Mhz B1MHz: bandpass, 1-MHz BW B10Mhz: bandpass, 10-MHz BW GAUSs: Gaussian, 100-kHz BW NYQuist: Nyquist, 100-kHz BW NY1Mhz: Nyquist, 1-MHz BW

get_mlength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:MLENgth
value: float = driver.configure.gprf.measurement.iqVsSlot.get_mlength()
```

Sets the length of the evaluation intervals used to calculate the I/Q vs slot results for one measurement step.

return
meas_length: No help available

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:REPetition
value: enums.Repeat = driver.configure.gprf.measurement.iqVsSlot.get_
↪repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

return
repetition: SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SCOunt
value: int = driver.configure.gprf.measurement.iqVsSlot.get_scount()
```

Defines the number of steps (measurement intervals) per subsweep. In list mode, the total number of steps must not exceed 3000 (step count times number of subsweeps) .

return
step_count: No help available

get_slength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SLENgth
value: float = driver.configure.gprf.measurement.iqVsSlot.get_slength()
```

Sets the time between the beginning of two consecutive measurement steps.

return
step_length: No help available

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:TOUT
value: float = driver.configure.gprf.measurement.iqVsSlot.get_timeout()
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
tcd_timeout: No help available

set_fe_limit(*limit: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:FELimit
driver.configure.gprf.measurement.iqVsSlot.set_fe_limit(limit = 1.0)
```

Defines the frequency estimation limit as a signal level relative to the expected nominal power. Steps with a level below this limit are not used for the frequency correction and do not contribute to the frequency results.

param limit
Range-100 dB to 0 dB*RST-100 dBDefault unitdB

set_ftype(*filter_type: FilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:FTYPe
driver.configure.gprf.measurement.iqVsSlot.set_ftype(filter_type = enums.
↪FilterType.B10Mhz)
```

Selects the IF filter type.

param filter_type
IF unit: B1MHz | B10Mhz RF unit: B1MHz | B10Mhz R&S CMW: GAUSs | NYQuist | NY1Mhz B1MHz: bandpass, 1-MHz BW B10Mhz: bandpass, 10-MHz BW GAUSs: Gaussian, 100-kHz BW NYQuist: Nyquist, 100-kHz BW NY1Mhz: Nyquist, 1-MHz BW

set_mlength(*meas_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:MLENgth
driver.configure.gprf.measurement.iqVsSlot.set_mlength(meas_length = 1.0)
```

Sets the length of the evaluation intervals used to calculate the I/Q vs slot results for one measurement step.

param meas_length
No help available

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:REPetition
driver.configure.gprf.measurement.iqVsSlot.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

param repetition
SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*step_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SCOunt
driver.configure.gprf.measurement.iqVsSlot.set_scount(step_count = 1)
```

Defines the number of steps (measurement intervals) per subsweep. In list mode, the total number of steps must not exceed 3000 (step count times number of subsweeps) .

param step_count
No help available

set_slength(*step_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SLENgth
driver.configure.gprf.measurement.iqVsSlot.set_slength(step_length = 1.0)
```

Sets the time between the beginning of two consecutive measurement steps.

param step_length
No help available

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:TOUT
driver.configure.gprf.measurement.iqVsSlot.set_timeout(tcd_timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param tcd_timeout
No help available

Cloning the Group

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

Subgroups

6.4.1.2.6.1 ListPy

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STARt
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STOP
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:COUNt
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST
```

class ListPyCls

ListPy commands group definition. 11 total commands, 4 Subgroups, 4 group commands

get_count() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:COUNt
value: int = driver.configure.gprf.measurement.iqVsSlot.listPy.get_count()
```

Queries the number of subsweeps per sweep. The total number of steps must not exceed 3000 (step count times number of subsweeps) .

return
sweep_count: No help available

get_start() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STARt
value: int = driver.configure.gprf.measurement.iqVsSlot.listPy.get_start()
```

Selects the first subsweep to be measured. The <StartIndex> must not be greater than the <StopIndex>. The total number of steps must not exceed 3000 (step count times number of subsweeps) .

return
start_index: No help available

get_stop() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STOP
value: int = driver.configure.gprf.measurement.iqVsSlot.listPy.get_stop()
```

Selects the last subsweep to be measured. The <StopIndex> must not be smaller than the <StartIndex>. The total number of steps must not exceed 3000 (step count times number of subsweeps) .

return
stop_index: No help available

get_value() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST
value: bool = driver.configure.gprf.measurement.iqVsSlot.listPy.get_value()
```

Enables or disables the list mode for the I/Q vs slot measurement.

return
list_mode: OFF: list mode off ON: list mode on

set_start(*start_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STARt
driver.configure.gprf.measurement.iqVsSlot.listPy.set_start(start_index = 1)
```

Selects the first subsweep to be measured. The <StartIndex> must not be greater than the <StopIndex>. The total number of steps must not exceed 3000 (step count times number of subsweeps) .

param start_index
No help available

set_stop(*stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STOP
driver.configure.gprf.measurement.iqVsSlot.listPy.set_stop(stop_index = 1)
```

Selects the last subsweep to be measured. The <StopIndex> must not be smaller than the <StartIndex>. The total number of steps must not exceed 3000 (step count times number of subsweeps) .

param stop_index
No help available

set_value(*list_mode: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST
driver.configure.gprf.measurement.iqVsSlot.listPy.set_value(list_mode = False)
```

Enables or disables the list mode for the I/Q vs slot measurement.

param list_mode
OFF: list mode off ON: list mode on

Cloning the Group

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

Subgroups

6.4.1.2.6.2 EnvelopePower

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower:ALL
```

class EnvelopePowerCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower
value: float = driver.configure.gprf.measurement.iqVsSlot.listPy.envelopePower.
↪get(index = 1)
```

Defines or queries the expected nominal power of subsweep <Index>.

param index
No help available

return
exp_nom_power: The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower:ALL
value: List[float] = driver.configure.gprf.measurement.iqVsSlot.listPy.
↪envelopePower.get_all()
```

Defines the expected nominal power for all subsweeps.

return
exp_nom_power: Comma-separated list of expected powers, one value per subsweep The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

set(*index: int*, *exp_nom_power: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower
driver.configure.gprf.measurement.iqVsSlot.listPy.envelopePower.set(index = 1,␣
↪exp_nom_power = 1.0)
```

Defines or queries the expected nominal power of subsweep <Index>.

param index
No help available

param exp_nom_power
The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

`set_all`(*exp_nom_power: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower:ALL
driver.configure.gprf.measurement.iqVsSlot.listPy.envelopePower.set_all(exp_nom_
↪power = [1.1, 2.2, 3.3])
```

Defines the expected nominal power for all subsweeps.

param exp_nom_power
Comma-separated list of expected powers, one value per subsweep The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

6.4.1.2.6.3 Frequency

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency:ALL
```

`class FrequencyCls`

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

`get`(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency
value: float = driver.configure.gprf.measurement.iqVsSlot.listPy.frequency.
↪get(index = 1)
```

Defines or queries the frequency of subsweep <Index>. For the supported frequency range, see 'Frequency ranges'.

param index
No help available

return
frequency: No help available

`get_all`() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency:ALL
value: List[float] = driver.configure.gprf.measurement.iqVsSlot.listPy.
↪frequency.get_all()
```

Defines the frequencies for all subsweeps. For the supported frequency range, see 'Frequency ranges'.

return
frequency: Comma-separated list of frequencies, one value per subsweep

set(*index: int*, *frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency
driver.configure.gprf.measurement.iqVsSlot.listPy.frequency.set(index = 1,␣
↪frequency = 1.0)
```

Defines or queries the frequency of subsweep <Index>. For the supported frequency range, see 'Frequency ranges'.

param index
No help available

param frequency
No help available

set_all(*frequency: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency:ALL
driver.configure.gprf.measurement.iqVsSlot.listPy.frequency.set_all(frequency =␣
↪[1.1, 2.2, 3.3])
```

Defines the frequencies for all subsweeps. For the supported frequency range, see 'Frequency ranges'.

param frequency
Comma-separated list of frequencies, one value per subsweep

6.4.1.2.6.4 Retrigger

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:RETRigger
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:RETRigger:ALL
```

class RetriggerCls

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

get(*index: int*) → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:RETRigger
value: bool = driver.configure.gprf.measurement.iqVsSlot.listPy.retrigger.
↪get(index = 1)
```

Configures the retrigger mechanism for subsweep <Index>. The setting is only relevant for trigger mode Retrigger Preselect.

param index
No help available

return
retrigger: No help available

get_all() → List[bool]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:RETRigger:ALL
value: List[bool] = driver.configure.gprf.measurement.iqVsSlot.listPy.retrigger.
↪get_all()
```

Configures the retrigger mechanism for all subsweeps. The setting is only relevant for trigger mode Retrigger Preselect.

return
retrigger: Comma-separated list of values, one value per subsweep

set(*index: int*, *retrigger: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:RETRigger
driver.configure.gprf.measurement.iqVsSlot.listPy.retrigger.set(index = 1,␣
↪retrigger = False)
```

Configures the retrigger mechanism for subsweep <Index>. The setting is only relevant for trigger mode Retrigger Preselect.

param index
No help available

param retrigger
No help available

set_all(*retrigger: List[bool]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:RETRigger:ALL
driver.configure.gprf.measurement.iqVsSlot.listPy.retrigger.set_all(retrigger =␣
↪[True, False, True])
```

Configures the retrigger mechanism for all subsweeps. The setting is only relevant for trigger mode Retrigger Preselect.

param retrigger
Comma-separated list of values, one value per subsweep

6.4.1.2.6.5 Sstop

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:SSTop
```

class SstopCls

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

class SstopStruct

Response structure. Fields:

- Start_Index: int: No parameter help available
- Stop_Index: int: No parameter help available

get() → SstopStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:SSTop
value: SstopStruct = driver.configure.gprf.measurement.iqVsSlot.listPy.sstop.
↪get()
```

Selects the range of subsweeps to be measured (first and last subsweep of a sweep) . The total number of steps must not exceed 3000 (step count times number of subsweeps) .

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

set(*start_index: int*, *stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:SSTop
driver.configure.gprf.measurement.iqVsSlot.listPy.sstop.set(start_index = 1,␣
↪stop_index = 1)
```

Selects the range of subsweeps to be measured (first and last subsweep of a sweep) . The total number of steps must not exceed 3000 (step count times number of subsweeps) .

param start_index
No help available

param stop_index
No help available

6.4.1.2.6.6 Trigger

SCPI Command :

```
[CONFigure]:GPRF:MEASurement<Instance>:IQVSlot:TRIGger:SOURce
```

class TriggerCls

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

get_source() → TriggerSource

```
# SCPI: [CONFigure]:GPRF:MEASurement<Instance>:IQVSlot:TRIGger:SOURce
value: enums.TriggerSource = driver.configure.gprf.measurement.iqVsSlot.trigger.
↪get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: TriggerSource*) → None

```
# SCPI: [CONFigure]:GPRF:MEASurement<Instance>:IQVSlot:TRIGger:SOURce
driver.configure.gprf.measurement.iqVsSlot.trigger.set_source(trigger = enums.
↪TriggerSource.EXTernal)
```

No command help available

param trigger
No help available

6.4.1.2.7 Nrpm

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:NRPM:SCOunt
CONFigure:GPRF:MEASurement<Instance>:NRPM:REPetition
CONFigure:GPRF:MEASurement<Instance>:NRPM:TOUT
```

class NrpmCls

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

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:REPetition
value: enums.Repeat = driver.configure.gprf.measurement.nrpm.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

return
repetition: SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:SCOunt
value: int = driver.configure.gprf.measurement.nrpm.get_scount()
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

return
statistic_count: Number of measurement intervals

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:TOUT
value: float = driver.configure.gprf.measurement.nrpm.get_timeout()
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
tcd_timeout: No help available

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:REPetition
driver.configure.gprf.measurement.nrpm.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

param repetition
SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:SCOunt
driver.configure.gprf.measurement.nrpm.set_scount(statistic_count = 1)
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

param statistic_count
Number of measurement intervals

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:TOUT
driver.configure.gprf.measurement.nrpm.set_timeout(tcd_timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param tcd_timeout
No help available

Cloning the Group

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

Subgroups

6.4.1.2.7.1 Sensor<Sensor>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.gprf.measurement.nrpm.sensor.repcap_sensor_get()
driver.configure.gprf.measurement.nrpm.sensor.repcap_sensor_set(repcap.Sensor.Nr1)
```

class SensorCls

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

Cloning the Group

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

Subgroups

6.4.1.2.7.2 Frequency

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:FREQuency
```

class FrequencyCls

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

get(*sensor=Sensor.Default*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:FREQuency
value: float = driver.configure.gprf.measurement.nrpm.sensor.frequency.
↪get(sensor = repcap.Sensor.Default)
```

Specifies the input frequency at the power sensor connected to Sensor <no>.

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

return
frequency: No help available

set(*frequency: float*, *sensor=Sensor.Default*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:FREQuency
driver.configure.gprf.measurement.nrpm.sensor.frequency.set(frequency = 1.0,␣
↪sensor = repcap.Sensor.Default)
```

Specifies the input frequency at the power sensor connected to Sensor <no>.

param frequency
No help available

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

6.4.1.2.8 Ploss

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:PLOSs:TRACe
CONFigure:GPRF:MEASurement<Instance>:PLOSs:MMODe
```

class PlossCls

Ploss commands group definition. 8 total commands, 3 Subgroups, 2 group commands

get_mmode() → MeasMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:MMODe
value: enums.MeasMode = driver.configure.gprf.measurement.ploss.get_mmode()
```

Selects the measurement mode.

return
meas_mode: OSHort: measurement with an 'open' and with a 'short' OOPen: measurement with an 'open'

get_trace() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TRACe
value: bool = driver.configure.gprf.measurement.ploss.get_trace()
```

Selects whether a result diagram is provided.

return
trace_mode: OFF: no result diagram, faster measurement ON: result diagram, slower measurement

set_mmode(*meas_mode: MeasMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:MMODe
driver.configure.gprf.measurement.ploss.set_mmode(meas_mode = enums.MeasMode.
↪CCALibration)
```

Selects the measurement mode.

param meas_mode
OSHort: measurement with an 'open' and with a 'short' OOPen: measurement with an 'open'

set_trace(*trace_mode: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TRACe
driver.configure.gprf.measurement.ploss.set_trace(trace_mode = False)
```

Selects whether a result diagram is provided.

param trace_mode
OFF: no result diagram, faster measurement ON: result diagram, slower measurement

Cloning the Group

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

Subgroups

6.4.1.2.8.1 ListPy

class ListPyCls

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

Cloning the Group

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

Subgroups

6.4.1.2.8.2 Frequency

SCPI Command :

```
CONFigure:GPRF:MEASurement<instance>:PLOSs:LIST:FREQuency
```

class FrequencyCls

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

class GetStruct

Response structure. Fields:

- Num_Entries: int: Configures the number of frequencies to be defined.
- Frequency: List[float]: Comma-separated list of NumEntries frequencies.

get(*connection_name: str*) → GetStruct

```
# SCPI: CONFigure:GPRF:MEASurement<instance>:PLOSs:LIST:FREQuency
value: GetStruct = driver.configure.gprf.measurement.ploss.listPy.frequency.
↪get(connection_name = 'abc')
```

Configures the frequency list for a selected RF connection. For possible connection names, see method RsCMPX_Gprf.Catalog. Gprf.Measurement.Ploss.cname. For the supported frequency range, see 'Frequency ranges'.

param connection_name
RF connection for which the frequency list is configured.

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

set(*connection_name: str*, *num_entries: int*, *frequency: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<instance>:PLOSs:LIST:FREQuency
driver.configure.gprf.measurement.ploss.listPy.frequency.set(connection_name =
↪'abc', num_entries = 1, frequency = [1.1, 2.2, 3.3])
```

Configures the frequency list for a selected RF connection. For possible connection names, see method RsCMPX_Gprf.Catalog. Gprf.Measurement.Ploss.cname. For the supported frequency range, see 'Frequency ranges'.

param connection_name
RF connection for which the frequency list is configured.

param num_entries
Configures the number of frequencies to be defined.

param frequency
Comma-separated list of NumEntries frequencies.

6.4.1.2.8.3 TsTone

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:ENABle
```

class TsToneCls

TsTone commands group definition. 4 total commands, 1 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:ENABle
value: bool = driver.configure.gprf.measurement.ploss.tsTone.get_enable()
```

Enables using configured Touchstone files.

return
enable: No help available

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:ENABle
driver.configure.gprf.measurement.ploss.tsTone.set_enable(enable = False)
```

Enables using configured Touchstone files.

param enable
No help available

Cloning the Group

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

Subgroups

6.4.1.2.8.4 File

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:OPEN
CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:SHORt
CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:MATCh
```

class FileCls

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

get_match() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:MATCh
value: str = driver.configure.gprf.measurement.ploss.tsTone.file.get_match()
```

No command help available

return
filename: No help available

get_open() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:OPEN
value: str = driver.configure.gprf.measurement.ploss.tsTone.file.get_open()
```

Selects a Touchstone file characterizing the 'open'.

return
filename: Path and filename, e.g. '@USERDATA/MyOpenFile.s1p'

get_short() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:SHORt
value: str = driver.configure.gprf.measurement.ploss.tsTone.file.get_short()
```

Selects a Touchstone file characterizing the 'short'.

return
filename: Path and filename, e.g. '@USERDATA/MyShortFile.s1p'

set_match(*filename: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:MATCh
driver.configure.gprf.measurement.ploss.tsTone.file.set_match(filename = 'abc')
```

No command help available

param filename
No help available

set_open(*filename: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:OPEN
driver.configure.gprf.measurement.ploss.tsTone.file.set_open(filename = 'abc')
```

Selects a Touchstone file characterizing the 'open'.

param filename
Path and filename, e.g. '@USERDATA/MyOpenFile.s1p'

set_short(*filename: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TSTone:FILE:SHORt
driver.configure.gprf.measurement.ploss.tsTone.file.set_short(filename = 'abc')
```

Selects a Touchstone file characterizing the 'short'.

param filename
Path and filename, e.g. '@USERDATA/MyShortFile.s1p'

6.4.1.2.8.5 View

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:PLOSs:VIEW:AFTaps
```

class ViewCls

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

get_aftaps() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:VIEW:AFTaps
value: int = driver.configure.gprf.measurement.ploss.view.get_aftaps()
```

Configures the number of frequencies over which the gain results are averaged.

return
avg_filter_taps: No help available

set_aftaps(*avg_filter_taps: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:VIEW:AFTaps
driver.configure.gprf.measurement.ploss.view.set_aftaps(avg_filter_taps = 1)
```

Configures the number of frequencies over which the gain results are averaged.

param avg_filter_taps
No help available

6.4.1.2.9 Power

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:MODE
CONFigure:GPRF:MEASurement<Instance>:POWer:TOUT
CONFigure:GPRF:MEASurement<Instance>:POWer:SLENgth
CONFigure:GPRF:MEASurement<Instance>:POWer:MLENgth
CONFigure:GPRF:MEASurement<Instance>:POWer:REPetition
CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
CONFigure:GPRF:MEASurement<Instance>:POWer:PDEFset
```

class PowerCls

Power commands group definition. 54 total commands, 5 Subgroups, 7 group commands

get_mlength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MLENgth
value: float = driver.configure.gprf.measurement.power.get_mlength()
```

Sets the length of the evaluation interval used to measure a single set of current power results. The measurement length cannot be greater than the step length.

return
meas_length: No help available

get_mode() → CcdfMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MODE
value: enums.CcdfMode = driver.configure.gprf.measurement.power.get_mode()
```

Selects the measurement mode for measurements without list mode. Select the mode before starting the power measurement.

return
ccdf_mode: No help available

get_pdef_set() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PDEFset
value: str = driver.configure.gprf.measurement.power.get_pdef_set()
```

No command help available

return
predefined_set: No help available

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:REPetition
value: enums.Repeat = driver.configure.gprf.measurement.power.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

return
repetition: SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
value: int = driver.configure.gprf.measurement.power.get_scount()
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

return
statistic_count: Number of measurement intervals

get_slength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SLENgth
value: float = driver.configure.gprf.measurement.power.get_slength()
```

Sets the time between the beginning of two consecutive measurement lengths.

return
step_length: No help available

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:TOUT
value: float = driver.configure.gprf.measurement.power.get_timeout()
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
tcd_timeout: No help available

set_mlength(*meas_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MLENgth
driver.configure.gprf.measurement.power.set_mlength(meas_length = 1.0)
```

Sets the length of the evaluation interval used to measure a single set of current power results. The measurement length cannot be greater than the step length.

param meas_length
No help available

set_mode(*ccdf_mode: CcdfMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MODE
driver.configure.gprf.measurement.power.set_mode(ccdf_mode = enums.CcdfMode.
↪POWer)
```

Selects the measurement mode for measurements without list mode. Select the mode before starting the power measurement.

param ccdf_mode
POWer: Power mode STATistic: Statistic mode

set_pdef_set(*predefined_set: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PDEFset
driver.configure.gprf.measurement.power.set_pdef_set(predefined_set = 'abc')
```

No command help available

param predefined_set
No help available

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:REPetition
driver.configure.gprf.measurement.power.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single shot or repeated continuously. Use CONFigure:..:MEAS<i>:...:SCOunt to determine the number of measurement intervals per single shot.

param repetition
SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
driver.configure.gprf.measurement.power.set_scount(statistic_count = 1)
```

Specifies the statistic count of the measurement. The statistic count is equal to the number of measurement intervals per single shot.

param statistic_count
Number of measurement intervals

set_slength(*step_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SLENgth
driver.configure.gprf.measurement.power.set_slength(step_length = 1.0)
```

Sets the time between the beginning of two consecutive measurement lengths.

param step_length
No help available

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:TOUT
driver.configure.gprf.measurement.power.set_timeout(tcd_timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement

is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param tcd_timeout
No help available

Cloning the Group

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

Subgroups

6.4.1.2.9.1 Catalog

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:CATalog:PDEFset
```

class CatalogCls

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

get_pdef_set() → List[str]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:CATalog:PDEFset
value: List[str] = driver.configure.gprf.measurement.power.catalog.get_pdef_
↪set()
```

No command help available

return
predefined_set: No help available

6.4.1.2.9.2 FilterPy

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:TYPE
```

class FilterPyCls

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

get_type_py() → DigitalFilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:TYPE
value: enums.DigitalFilterType = driver.configure.gprf.measurement.power.
↪filterPy.get_type_py()
```

Selects the IF filter type.

return
filter_type: RF unit: BANDpass | GAUSs IF unit: BANDpass | GAUSs R&S CMW: BANDpass | GAUSs | WCDMa | CDMA | TDSCdma BANDpass: bandpass filter GAUSs: Gaussian filter WCDMA: 3.84-MHz RRC filter for WCDMA TX tests CDMA: 1.2288-MHz channel filter for CDMA 2000 TX tests TDSCdma: 1.28-MHz RRC filter for TD-SCDMA TX tests

set_type_py(*filter_type: DigitalFilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:TYPE
driver.configure.gprf.measurement.power.filterPy.set_type_py(filter_type =␣
↪enums.DigitalFilterType.BANDpass)
```

Selects the IF filter type.

param filter_type
RF unit: BANDpass | GAUSs IF unit: BANDpass | GAUSs R&S CMW: BANDpass | GAUSs | WCDMa | CDMA | TDSCdma BANDpass: bandpass filter GAUSs: Gaussian filter WCDMA: 3.84-MHz RRC filter for WCDMA TX tests CDMA: 1.2288-MHz channel filter for CDMA 2000 TX tests TDSCdma: 1.28-MHz RRC filter for TD-SCDMA TX tests

Cloning the Group

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

Subgroups

6.4.1.2.9.3 Bandpass

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:BANDpass:BWIDth
```

class BandpassCls

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

get_bandwidth() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:BANDpass:BWIDth
value: float = driver.configure.gprf.measurement.power.filterPy.bandpass.get_
↪bandwidth()
```

Selects the bandwidth for a bandpass filter.

return
bandpass_bw: For allowed values, see Table ‘Supported values’.

set_bandwidth(*bandpass_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:BANDpass:BWIDth
driver.configure.gprf.measurement.power.filterPy.bandpass.set_
↪bandwidth(bandpass_bw = 1.0)
```

Selects the bandwidth for a bandpass filter.

param bandpass_bw
For allowed values, see Table 'Supported values'.

6.4.1.2.9.4 Gauss

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:GAUSs:BWIDth
```

class GaussCls

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

get_bandwidth() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:GAUSs:BWIDth
value: float = driver.configure.gprf.measurement.power.filterPy.gauss.get_
↪bandwidth()
```

Selects the bandwidth for a filter of Gaussian shape.

return
gauss_bw: For allowed values, see Table 'Supported values'.

set_bandwidth(*gauss_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:GAUSs:BWIDth
driver.configure.gprf.measurement.power.filterPy.gauss.set_bandwidth(gauss_bw =␣
↪1.0)
```

Selects the bandwidth for a filter of Gaussian shape.

param gauss_bw
For allowed values, see Table 'Supported values'.

6.4.1.2.9.5 ListPy

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXIMode
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXITiming
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FILL
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:MUNit
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:COUNt
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STARt
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STOP
[CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:CSOurce
```

(continues on next page)

(continued from previous page)

```
[CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:NIDX
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST
```

class ListPyCls

ListPy commands group definition. 27 total commands, 9 Subgroups, 10 group commands

class FillStruct

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

- Start_Index: float: No parameter help available
- Range_Py: float: No parameter help available
- Index_Repetition: float: No parameter help available
- Start_Frequency: float: No parameter help available
- Freq_Increment: float: No parameter help available
- Start_Power: float: No parameter help available
- Power_Increment: float: No parameter help available
- Retrigger: bool: No parameter help available
- Iq_Data: bool: No parameter help available
- Parameter_Set: int: No parameter help available

get_count() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:COUNt
value: int = driver.configure.gprf.measurement.power.listPy.get_count()
```

Queries the total number of segments per sweep, including repetitions.

return
result_count: No help available

get_csource() → ConnectionSource

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:CSOurce
value: enums.ConnectionSource = driver.configure.gprf.measurement.power.listPy.
↪get_csource()
```

Selects whether all list mode segments use the same RF connection.

return
connection_source: GLOBal: Use the same RF connection for all segments. INDex: Assign a connection index to each segment.

get_munit() → MagnitudeUnit

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:MUNit
value: enums.MagnitudeUnit = driver.configure.gprf.measurement.power.listPy.get_
↪munit()
```

No command help available

return
magnitude_unit: No help available

get_nidx() → int

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:NIDX
value: int = driver.configure.gprf.measurement.power.listPy.get_nidx()
```

Sets the number of connection indices for the list mode, for the connection source INDex.

return
number_of_indices: No help available

get_start() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STARt
value: int = driver.configure.gprf.measurement.power.listPy.get_start()
```

Selects the first segment to be measured (start of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

return
start_index: No help available

get_stop() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STOP
value: int = driver.configure.gprf.measurement.power.listPy.get_stop()
```

Selects the last segment to be measured (the end of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

return
stop_index: No help available

get_txi_mode() → TxiMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXIMode
value: enums.TxiMode = driver.configure.gprf.measurement.power.listPy.get_txi_
↪mode()
```

Selects the repetition of the GPRF Meas<i>:Power trigger signal, generated by the power measurement for a repeated segment.

return
mode: - IREPetition: Index repetition - trigger signal after each repetition of the segment. - LENTry: List entry - trigger signal only after the last repetition of the segment.

get_txi_timing() → Timing

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXITiming
value: enums.Timing = driver.configure.gprf.measurement.power.listPy.get_txi_
↪timing()
```

Specifies the timing of the generated GPRF Meas<i>:Power trigger.

return
timing: STEP: Trigger signals are generated between step lengths. CENTered: Trigger signals are generated between measurement lengths.

get_value() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST
value: bool = driver.configure.gprf.measurement.power.listPy.get_value()
```

Enables or disables the list mode for the power measurement.

return
enable_list_mode: OFF: list mode off ON: list mode on

set_csource(*connection_source: ConnectionSource*) → None

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:CSOurce
driver.configure.gprf.measurement.power.listPy.set_csource(connection_source =␣
↪enums.ConnectionSource.GLOBal)
```

Selects whether all list mode segments use the same RF connection.

param connection_source
GLOBal: Use the same RF connection for all segments. INDex: Assign a connection index to each segment.

set_fill(*value: FillStruct*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FILL
structure = driver.configure.gprf.measurement.power.listPy.FillStruct()
structure.Start_Index: float = 1.0
structure.Range_Py: float = 1.0
structure.Index_Repetition: float = 1.0
structure.Start_Frequency: float = 1.0
structure.Freq_Increment: float = 1.0
structure.Start_Power: float = 1.0
structure.Power_Increment: float = 1.0
structure.Retrigger: bool = False
structure.Iq_Data: bool = False
structure.Parameter_Set: int = 1
driver.configure.gprf.measurement.power.listPy.set_fill(value = structure)
```

No command help available

param value
see the help for FillStruct structure arguments.

set_munit(*magnitude_unit: MagnitudeUnit*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:MUNit
driver.configure.gprf.measurement.power.listPy.set_munit(magnitude_unit = enums.
↪MagnitudeUnit.RAW)
```

No command help available

param magnitude_unit
No help available

set_nidx(*number_of_indices: int*) → None

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:NIDX
driver.configure.gprf.measurement.power.listPy.set_nidx(number_of_indices = 1)
```

Sets the number of connection indices for the list mode, for the connection source INDex.

param number_of_indices
No help available

set_start(*start_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STARt
driver.configure.gprf.measurement.power.listPy.set_start(start_index = 1)
```

Selects the first segment to be measured (start of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

param start_index
No help available

set_stop(*stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STOP
driver.configure.gprf.measurement.power.listPy.set_stop(stop_index = 1)
```

Selects the last segment to be measured (the end of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

param stop_index
No help available

set_txi_mode(*mode: TxiMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXIMode
driver.configure.gprf.measurement.power.listPy.set_txi_mode(mode = enums.
↪TxiMode.IREPetition)
```

Selects the repetition of the GPRF Meas<i>:Power trigger signal, generated by the power measurement for a repeated segment.

param mode

- IREPetition: Index repetition - trigger signal after each repetition of the segment.
- LENTry: List entry - trigger signal only after the last repetition of the segment.

set_txi_timing(*timing: Timing*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXITiming
driver.configure.gprf.measurement.power.listPy.set_txi_timing(timing = enums.
↪Timing.CENTered)
```

Specifies the timing of the generated GPRF Meas<i>:Power trigger.

param timing
STEP: Trigger signals are generated between step lengths. CENTered: Trigger signals are generated between measurement lengths.

set_value(*enable_list_mode: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST
driver.configure.gprf.measurement.power.listPy.set_value(enable_list_mode =␣
↪False)
```

Enables or disables the list mode for the power measurement.

param enable_list_mode
OFF: list mode off ON: list mode on

Cloning the Group

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

Subgroups

6.4.1.2.9.6 Cidx

SCPI Command :

```
[CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:CIDX
```

class CidxCls

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

class CidxStruct

Response structure. Fields:

- Row_Number: int: No parameter help available
- Connection_Index: int: No parameter help available

get() → CidxStruct

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:CIDX
value: CidxStruct = driver.configure.gprf.measurement.power.listPy.cidx.get()
```

Selects the RF connection index for segment <RowNumber>. For the definition of the connection indices, see [CONFigure:]GPRF:MEAS<i>:POWer:LIST:IDX<idx>:CONNection.

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

set(*row_number: int*, *connection_index: int*) → None

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:CIDX
driver.configure.gprf.measurement.power.listPy.cidx.set(row_number = 1,␣
↪connection_index = 1)
```

Selects the RF connection index for segment <RowNumber>. For the definition of the connection indices, see [CONFigure:]GPRF:MEAS<i>:POWer:LIST:IDX<idx>:CONNection.

param row_number
No help available

param connection_index
No help available

6.4.1.2.9.7 EnvelopePower

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower:ALL
```

class EnvelopePowerCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower
value: float = driver.configure.gprf.measurement.power.listPy.envelopePower.
↪get(index = 1)
```

Defines or queries the expected nominal power of segment <Index>.

param index
No help available

return
exp_nom_power: The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower:ALL
value: List[float] = driver.configure.gprf.measurement.power.listPy.
↪envelopePower.get_all()
```

Defines the expected nominal power for all segments.

return
exp_nom_power: Comma-separated list of expected powers, one value per segment The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

set(*index: int*, *exp_nom_power: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower
driver.configure.gprf.measurement.power.listPy.envelopePower.set(index = 1, exp_
↪nom_power = 1.0)
```

Defines or queries the expected nominal power of segment <Index>.

param index
No help available

param exp_nom_power
The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

set_all(*exp_nom_power: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower:ALL
driver.configure.gprf.measurement.power.listPy.envelopePower.set_all(exp_nom_
↪power = [1.1, 2.2, 3.3])
```

Defines the expected nominal power for all segments.

param exp_nom_power
Comma-separated list of expected powers, one value per segment The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

6.4.1.2.9.8 Frequency

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency:ALL
```

class FrequencyCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency
value: float = driver.configure.gprf.measurement.power.listPy.frequency.
↪get(index = 1)
```

Defines or queries the frequency of segment <Index>. For the supported frequency range, see 'Frequency ranges'.

param index
No help available

return
frequency: No help available

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency:ALL
value: List[float] = driver.configure.gprf.measurement.power.listPy.frequency.
↪get_all()
```

Defines the frequencies for all segments. For the supported frequency range, see 'Frequency ranges'.

return
frequency: Comma-separated list of frequencies, one value per segment

set(*index: int*, *frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency
driver.configure.gprf.measurement.power.listPy.frequency.set(index = 1,␣
↪frequency = 1.0)
```

Defines or queries the frequency of segment <Index>. For the supported frequency range, see 'Frequency ranges'.

param index
No help available

param frequency
No help available

`set_all`(*frequency: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency:ALL
driver.configure.gprf.measurement.power.listPy.frequency.set_all(frequency = [1.
↪1, 2.2, 3.3])
```

Defines the frequencies for all segments. For the supported frequency range, see 'Frequency ranges'.

param frequency
Comma-separated list of frequencies, one value per segment

6.4.1.2.9.9 Idx<Index>

RepCap Settings

```
# Range: Ix1 .. Ix32
rc = driver.configure.gprf.measurement.power.listPy.idx.repcap_index_get()
driver.configure.gprf.measurement.power.listPy.idx.repcap_index_set(repcap.Index.Ix1)
```

`class IdxCls`

Idx commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Index, default value after init: Index.Ix1

Cloning the Group

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

Subgroups

6.4.1.2.9.10 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.configure.gprf.measurement.power.listPy.idx.catalog.clone()
```

Subgroups

6.4.1.2.9.11 Connection

SCPI Command :

```
[CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:IDX<idx>:CATalog:CONNection
```

class ConnectionCls

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

get(*index=Index.Default*) → List[str]

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:IDX<idx>
↪:CATalog:CONNection
value: List[str] = driver.configure.gprf.measurement.power.listPy.idx.catalog.
↪connection.get(index = repcap.Index.Default)
```

Returns the names of the available connections.

param index
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Idx')

return
connection_source: Comma-separated list of strings, one string per connection

6.4.1.2.9.12 Connection

SCPI Command :

```
[CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:IDX<idx>:CONNection
```

class ConnectionCls

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

get(*index=Index.Default*) → str

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:IDX<idx>:CONNection
value: str = driver.configure.gprf.measurement.power.listPy.idx.connection.
↪get(index = repcap.Index.Default)
```

Assigns a connection to the connection index <idx>. Alternatively, use method RsCMPX_Gprf.Route.Gprf.Measurement.Spath. value. For possible <Connection> strings, see [CONFigure:]GPRF:MEAS<i>:POWer:LIST:IDX<idx>:CATalog:CONNection?.

param index
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Idx')

return
connection: No help available

set(*connection: str*, *index=Index.Default*) → None

```
# SCPI: [CONFigure]:GPRF:MEASurement<instance>:POWer:LIST:IDX<idx>:CONNection
driver.configure.gprf.measurement.power.listPy.idx.connection.set(connection =
↪'abc', index = repcap.Index.Default)
```

Assigns a connection to the connection index <idx>. Alternatively, use method RsCMPX_Gprf.Route.Gprf.Measurement.Spath. value. For possible <Connection> strings, see [CONFigure:]GPRF:MEAS<i>:POWer:LIST:IDX<idx>:CATalog:CONNection?.

param connection
No help available

param index
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Idx')

6.4.1.2.9.13 IqData

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData:CAPTure
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData:ALL
```

class IqDataCls

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

get(*index: int*) → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData
value: bool = driver.configure.gprf.measurement.power.listPy.iqData.get(index =␣
↪1)
```

No command help available

param index
No help available

return
iq_data: No help available

get_all() → List[bool]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData:ALL
value: List[bool] = driver.configure.gprf.measurement.power.listPy.iqData.get_
↪all()
```

No command help available

return
iq_data: No help available

get_capture() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData:CAPTure
value: bool = driver.configure.gprf.measurement.power.listPy.iqData.get_
↪capture()
```

No command help available

return
capture_iq_data: No help available

set(*index: int*, *iq_data: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData
driver.configure.gprf.measurement.power.listPy.iqData.set(index = 1, iq_data =␣
↪False)
```

No command help available

param index
No help available

param iq_data
No help available

set_all(*iq_data: List[bool]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData:ALL
driver.configure.gprf.measurement.power.listPy.iqData.set_all(iq_data = [True,␣
↪False, True])
```

No command help available

param iq_data
No help available

set_capture(*capture_iq_data: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IQData:CAPTure
driver.configure.gprf.measurement.power.listPy.iqData.set_capture(capture_iq_
↪data = False)
```

No command help available

param capture_iq_data
No help available

6.4.1.2.9.14 Irepetition

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition:ALL
```

class IrepetitionCls

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

get(*index: int*) → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition
value: int = driver.configure.gprf.measurement.power.listPy.irepetition.
↪get(index = 1)
```

Configures the number of repetitions of segment <Index>. The total number of repetitions over all measured segments must not be higher than 10000.

param index
No help available

return
repetition: No help available

get_all() → List[int]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition:ALL
value: List[int] = driver.configure.gprf.measurement.power.listPy.irepetition.
↪get_all()
```

Configures the number of repetitions for all segments. The total number of repetitions over all measured segments must not be higher than 10000.

return
repetition: Comma-separated list of repetitions, one value per segment

set(*index: int*, *repetition: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition
driver.configure.gprf.measurement.power.listPy.irepetition.set(index = 1,␣
↪repetition = 1)
```

Configures the number of repetitions of segment <Index>. The total number of repetitions over all measured segments must not be higher than 10000.

param index
No help available

param repetition
No help available

set_all(*repetition: List[int]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition:ALL
driver.configure.gprf.measurement.power.listPy.irepetition.set_all(repetition =␣
↪[1, 2, 3])
```

Configures the number of repetitions for all segments. The total number of repetitions over all measured segments must not be higher than 10000.

param repetition
Comma-separated list of repetitions, one value per segment

6.4.1.2.9.15 ParameterSetList

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET:ALL
```

class ParameterSetListCls

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

get(*index: int*) → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET
value: int = driver.configure.gprf.measurement.power.listPy.parameterSetList.
↪get(index = 1)
```

Selects the parameter set for segment <Index>.

param index
No help available

return
parameter_set: No help available

get_all() → List[int]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET:ALL
value: List[int] = driver.configure.gprf.measurement.power.listPy.
↪parameterSetList.get_all()
```

Selects the parameter set for all segments.

return
parameter_set: Comma-separated list of parameter set numbers, one value per segment.

set(*index: int*, *parameter_set: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET
driver.configure.gprf.measurement.power.listPy.parameterSetList.set(index = 1,␣
↪parameter_set = 1)
```

Selects the parameter set for segment <Index>.

param index
No help available

param parameter_set
No help available

set_all(*parameter_set: List[int]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET:ALL
driver.configure.gprf.measurement.power.listPy.parameterSetList.set_
↪all(parameter_set = [1, 2, 3])
```

Selects the parameter set for all segments.

param parameter_set
Comma-separated list of parameter set numbers, one value per segment.

6.4.1.2.9.16 Retrigger

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger:ALL
```

class RetriggerCls

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

get(*index: int*) → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger
value: bool = driver.configure.gprf.measurement.power.listPy.retrigger.
↪get(index = 1)
```

Configures the retrigger mechanism for segment <Index>. The setting is only relevant for trigger mode Retrigger Preselect.

param index
No help available

return
retrigger: No help available

get_all() → List[bool]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger:ALL
value: List[bool] = driver.configure.gprf.measurement.power.listPy.retrigger.
↪get_all()
```

Configures the retrigger mechanism for all segments. The setting is only relevant for trigger mode Retrigger Preselect.

return
retrigger: Comma-separated list of values, one value per segment

set(*index: int*, *retrigger: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger
driver.configure.gprf.measurement.power.listPy.retrigger.set(index = 1,␣
↪retrigger = False)
```

Configures the retrigger mechanism for segment <Index>. The setting is only relevant for trigger mode Retrigger Preselect.

param index
No help available

param retrigger
No help available

set_all(*retrigger: List[bool]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger:ALL
driver.configure.gprf.measurement.power.listPy.retrigger.set_all(retrigger =␣
↪[True, False, True])
```

Configures the retrigger mechanism for all segments. The setting is only relevant for trigger mode Retrigger Preselect.

param retrigger
Comma-separated list of values, one value per segment

6.4.1.2.9.17 Sstop

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:SSTop
```

class SstopCls

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

class SstopStruct

Response structure. Fields:

- Start_Index: int: No parameter help available
- Stop_Index: int: No parameter help available

get() → SstopStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:SSTop
value: SstopStruct = driver.configure.gprf.measurement.power.listPy.sstop.get()
```

Selects the range of segments to be measured (first and last segment of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

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

set(*start_index: int*, *stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:SSTop
driver.configure.gprf.measurement.power.listPy.sstop.set(start_index = 1, stop_
↪index = 1)
```

Selects the range of segments to be measured (first and last segment of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

param start_index
No help available

param stop_index
No help available

6.4.1.2.9.18 ParameterSetList

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET
```

class ParameterSetListCls

ParameterSetList commands group definition. 15 total commands, 5 Subgroups, 1 group commands

get_value() → ParameterSetMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET
value: enums.ParameterSetMode = driver.configure.gprf.measurement.power.
↪parameterSetList.get_value()
```

Selects whether all segments use the same measurement control settings.

return
parameter_set_mode: GLOBal: Use global settings for all segments. LIST: Use segment-specific settings.

set_value(*parameter_set_mode: ParameterSetMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET
driver.configure.gprf.measurement.power.parameterSetList.set_value(parameter_
↪set_mode = enums.ParameterSetMode.GLOBal)
```

Selects whether all segments use the same measurement control settings.

param parameter_set_mode
GLOBal: Use global settings for all segments. LIST: Use segment-specific settings.

Cloning the Group

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

Subgroups

6.4.1.2.9.19 Catalog

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:CATalog:PDEFset
```

class CatalogCls

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

get_pdef_set() → List[str]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:CATalog:PDEFset
value: List[str] = driver.configure.gprf.measurement.power.parameterSetList.
↪catalog.get_pdef_set()
```

Returns a comma-separated list of predefined parameter sets that can be loaded using method RsCMPX_Gprf.Configure.Gprf. Measurement.Power.ParameterSetList.PdefSet.set. See also 'Predefined parameter sets'.

return
predefined_set: Comma-separated list of strings

6.4.1.2.9.20 FilterPy

class FilterPyCls

FilterPy commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.1.2.9.21 Bandpass

class BandpassCls

Bandpass 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.gprf.measurement.power.parameterSetList.filterPy.bandpass.
↪clone()
```

Subgroups

6.4.1.2.9.22 Bandwidth

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BANDpass:BWIDth
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BANDpass:BWIDth:ALL
```

class BandwidthCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BANDpass:BWIDth
value: float = driver.configure.gprf.measurement.power.parameterSetList.
↪filterPy.bandpass.bandwidth.get(index = 1)
```

Selects the bandpass filter bandwidth for the parameter set <Index>.

param index
No help available

return
bandwidth: For supported values, see Table 'Supported values'.

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>
↪:POWer:PSET:FILTer:BANDpass:BWIDth:ALL
value: List[float] = driver.configure.gprf.measurement.power.parameterSetList.
↪filterPy.bandpass.bandwidth.get_all()
```

Selects the bandpass filter bandwidth for all parameter sets.

return
bandwidth: Comma-separated list of 32 values, for parameter set 0 to 31. For supported values, see Table 'Supported values'.

set(*index: int*, *bandwidth: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BANDpass:BWIDth
driver.configure.gprf.measurement.power.parameterSetList.filterPy.bandpass.
↪bandwidth.set(index = 1, bandwidth = 1.0)
```

Selects the bandpass filter bandwidth for the parameter set <Index>.

param index
No help available

param bandwidth
For supported values, see Table 'Supported values'.

set_all(*bandwidth: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>
↪:POWer:PSET:FILTer:BANDpass:BWIDth:ALL
driver.configure.gprf.measurement.power.parameterSetList.filterPy.bandpass.
↪bandwidth.set_all(bandwidth = [1.1, 2.2, 3.3])
```

Selects the bandpass filter bandwidth for all parameter sets.

param bandwidth
Comma-separated list of 32 values, for parameter set 0 to 31. For supported values, see Table 'Supported values'.

6.4.1.2.9.23 Bandwidth

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BWIDth
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BWIDth:ALL
```

class BandwidthCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BWIDth
value: float = driver.configure.gprf.measurement.power.parameterSetList.
↪filterPy.bandwidth.get(index = 1)
```

No command help available

param index
No help available

return
bandwidth: No help available

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BWIDth:ALL
value: List[float] = driver.configure.gprf.measurement.power.parameterSetList.
↪filterPy.bandwidth.get_all()
```

No command help available

return
bandwidth: No help available

set(*index: int*, *bandwidth: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BWIDth
driver.configure.gprf.measurement.power.parameterSetList.filterPy.bandwidth.
↪set(index = 1, bandwidth = 1.0)
```

No command help available

param index
No help available

param bandwidth
No help available

set_all(*bandwidth: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BWIDth:ALL
driver.configure.gprf.measurement.power.parameterSetList.filterPy.bandwidth.set_
↪all(bandwidth = [1.1, 2.2, 3.3])
```

No command help available

param bandwidth
No help available

6.4.1.2.9.24 Gauss

class GaussCls

Gauss 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.gprf.measurement.power.parameterSetList.filterPy.gauss.clone()
```

Subgroups

6.4.1.2.9.25 Bandwidth

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth:ALL
```

class BandwidthCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth
value: float = driver.configure.gprf.measurement.power.parameterSetList.
↪filterPy.gauss.bandwidth.get(index = 1)
```

Selects the bandwidth for a filter of Gaussian shape for the parameter set <Index>.

param index
No help available

return
bandwidth: For supported values, see Table 'Supported values'.

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth:ALL
value: List[float] = driver.configure.gprf.measurement.power.parameterSetList.
↪filterPy.gauss.bandwidth.get_all()
```

Selects the bandwidth for a filter of Gaussian shape for all parameter sets.

return
bandwidth: Comma-separated list of 32 values, for parameter set 0 to 31. For supported values, see Table 'Supported values'.

set(*index: int*, *bandwidth: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth
driver.configure.gprf.measurement.power.parameterSetList.filterPy.gauss.
↪bandwidth.set(index = 1, bandwidth = 1.0)
```

Selects the bandwidth for a filter of Gaussian shape for the parameter set <Index>.

param index
No help available

param bandwidth
For supported values, see Table 'Supported values'.

set_all(*bandwidth: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth:ALL
driver.configure.gprf.measurement.power.parameterSetList.filterPy.gauss.
↪bandwidth.set_all(bandwidth = [1.1, 2.2, 3.3])
```

Selects the bandwidth for a filter of Gaussian shape for all parameter sets.

param bandwidth
Comma-separated list of 32 values, for parameter set 0 to 31. For supported values, see Table 'Supported values'.

6.4.1.2.9.26 TypePy

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:TYPE
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:TYPE:ALL
```

class TypePyCls

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

get(*index: int*) → DigitalFilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:TYPE
value: enums.DigitalFilterType = driver.configure.gprf.measurement.power.
↪parameterSetList.filterPy.typePy.get(index = 1)
```

Selects the IF filter type for the parameter set <Index>.

param index
No help available

return
filter_py: RF unit: BANDpass | GAUSs IF unit: BANDpass | GAUSs R&S CMW: BANDpass | GAUSs | WCDMa | CDMA | TDSCdma BANDpass: bandpass filter GAUSs: Gaussian filter WCDMA: 3.84-MHz RRC filter for WCDMA TX tests CDMA: 1.2288-MHz channel filter for CDMA 2000 TX tests TDSCdma: 1.28-MHz RRC filter for TD-SCDMA TX tests

get_all() → List[DigitalFilterType]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:TYPE:ALL
value: List[enums.DigitalFilterType] = driver.configure.gprf.measurement.power.
↪parameterSetList.filterPy.typePy.get_all()
```

Selects the IF filter type for all parameter sets.

return
filter_py: RF unit: BANDpass | GAUSs IF unit: BANDpass | GAUSs R&S CMW: BANDpass | GAUSs | WCDMa | CDMA | TDSCdma Comma-separated list of 32 values, for parameter set 0 to 31. BANDpass: bandpass filter GAUSs: Gaussian filter WCDMA: 3.84-MHz RRC filter for WCDMA TX tests CDMA: 1.2288-MHz channel filter for CDMA 2000 TX tests TDSCdma: 1.28-MHz RRC filter for TD-SCDMA TX tests

set(*index: int*, *filter_py: DigitalFilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:TYPE
driver.configure.gprf.measurement.power.parameterSetList.filterPy.typePy.
↪set(index = 1, filter_py = enums.DigitalFilterType.BANDpass)
```

Selects the IF filter type for the parameter set <Index>.

param index
No help available

param filter_py
RF unit: BANDpass | GAUSs IF unit: BANDpass | GAUSs R&S CMW: BANDpass | GAUSs | WCDMa | CDMA | TDSCdma BANDpass: bandpass filter GAUSs: Gaussian filter WCDMA: 3.84-MHz RRC filter for WCDMA TX tests CDMA: 1.2288-MHz channel filter for CDMA 2000 TX tests TDSCdma: 1.28-MHz RRC filter for TD-SCDMA TX tests

set_all(*filter_py: List[DigitalFilterType]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:TYPE:ALL
driver.configure.gprf.measurement.power.parameterSetList.filterPy.typePy.set_
↪all(filter_py = [DigitalFilterType.BANDpass, DigitalFilterType.WCDMa])
```

Selects the IF filter type for all parameter sets.

param filter_py
RF unit: BANDpass | GAUSs IF unit: BANDpass | GAUSs R&S CMW: BANDpass | GAUSs | WCDMa | CDMA | TDSCdma Comma-separated list of 32 values, for parameter set 0 to 31. BANDpass: bandpass filter GAUSs: Gaussian filter WCDMA: 3.84-MHz RRC filter for WCDMA TX tests CDMA: 1.2288-MHz channel filter for CDMA 2000 TX tests TDSCdma: 1.28-MHz RRC filter for TD-SCDMA TX tests

6.4.1.2.9.27 Mlength

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth:ALL
```

class MlengthCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth
value: float = driver.configure.gprf.measurement.power.parameterSetList.mlength.
↪get(index = 1)
```

Sets the length of the evaluation interval used to measure a single set of current power results for the parameter set <Index>. The measurement length cannot be greater than the step length.

param index
No help available

return
meas_length: No help available

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth:ALL
value: List[float] = driver.configure.gprf.measurement.power.parameterSetList.
↪mlength.get_all()
```

Sets the length of the evaluation interval used to measure a single set of current power results, for all parameter sets. The measurement length cannot be greater than the step length.

return
meas_length: Comma-separated list of 32 values, for parameter set 0 to 31.

set(*index: int*, *meas_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth
driver.configure.gprf.measurement.power.parameterSetList.mlength.set(index = 1,␣
↪meas_length = 1.0)
```

Sets the length of the evaluation interval used to measure a single set of current power results for the parameter set <Index>. The measurement length cannot be greater than the step length.

param index
No help available

param meas_length
No help available

set_all(*meas_length: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth:ALL
driver.configure.gprf.measurement.power.parameterSetList.mlength.set_all(meas_
↪length = [1.1, 2.2, 3.3])
```

Sets the length of the evaluation interval used to measure a single set of current power results, for all parameter sets. The measurement length cannot be greater than the step length.

param meas_length
Comma-separated list of 32 values, for parameter set 0 to 31.

6.4.1.2.9.28 PdefSet

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:PDEFset
```

class PdefSetCls

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

get(*index: int*) → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:PDEFset
value: str = driver.configure.gprf.measurement.power.parameterSetList.pdefSet.
↪get(index = 1)
```

This command is related to parameter sets in retriggered list mode. A setting command loads a predefined set of parameters into the parameter set <Index>. A query returns the name of the predefined set assigned to the parameter set <Index>. To get a list of allowed strings for <PredefinedSet>, use method RsCMPX_Gprf.Configure.Gprf.Measurement.Power. ParameterSetList.Catalog.pdefSet.

param index
No help available

return
predefined_set: No help available

set(*index: int*, *predefined_set: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:PDEFset
driver.configure.gprf.measurement.power.parameterSetList.pdefSet.set(index = 1,␣
↪predefined_set = 'abc')
```

This command is related to parameter sets in retriggered list mode. A setting command loads a predefined set of parameters into the parameter set <Index>. A query returns the name of the predefined set assigned to the parameter set <Index>. To get a list of allowed strings for <PredefinedSet>, use method RsCMPX_Gprf.Configure.Gprf.Measurement.Power. ParameterSetList.Catalog.pdefSet.

param index
No help available

param predefined_set
No help available

6.4.1.2.9.29 Slength

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth:ALL
```

class SlengthCls

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

get(*index: int*) → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth
value: float = driver.configure.gprf.measurement.power.parameterSetList.slength.
↪get(index = 1)
```

Selects the time between the beginning of two consecutive measurement lengths for the parameter set <Index>.

param index
No help available

return
step_length: No help available

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth:ALL
value: List[float] = driver.configure.gprf.measurement.power.parameterSetList.
↪slength.get_all()
```

Selects the time between the beginning of two consecutive measurement lengths for all parameter sets.

return
step_length: Comma-separated list of 32 values, for parameter set 0 to 31.

set(*index: int*, *step_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth
driver.configure.gprf.measurement.power.parameterSetList.slength.set(index = 1,␣
↪step_length = 1.0)
```

Selects the time between the beginning of two consecutive measurement lengths for the parameter set <Index>.

param index
No help available

param step_length
No help available

set_all(*step_length: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth:ALL
driver.configure.gprf.measurement.power.parameterSetList.slength.set_all(step_
↪length = [1.1, 2.2, 3.3])
```

Selects the time between the beginning of two consecutive measurement lengths for all parameter sets.

param step_length
Comma-separated list of 32 values, for parameter set 0 to 31.

6.4.1.2.9.30 Trigger

SCPI Command :

```
[CONFigure]:GPRF:MEASurement<Instance>:POWer:TRIGger:SOURce
```

class TriggerCls

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

get_source() → TriggerSource

```
# SCPI: [CONFigure]:GPRF:MEASurement<Instance>:POWer:TRIGger:SOURce
value: enums.TriggerSource = driver.configure.gprf.measurement.power.trigger.
↪get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: TriggerSource*) → None

```
# SCPI: [CONFigure]:GPRF:MEASurement<Instance>:POWer:TRIGger:SOURce
driver.configure.gprf.measurement.power.trigger.set_source(trigger = enums.
↪TriggerSource.EXTernal)
```

No command help available

param trigger
No help available

6.4.1.2.10 RfSettings

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:RFSettings:LOFRequency
CONFigure:GPRF:MEASurement<Instance>:RFSettings:LOLevel
CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
CONFigure:GPRF:MEASurement<Instance>:RFSettings:ENPower
CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
CONFigure:GPRF:MEASurement<Instance>:RFSettings:UMARgin
CONFigure:GPRF:MEASurement<Instance>:RFSettings:MLOFfset
CONFigure:GPRF:MEASurement<Instance>:RFSettings:FOFFset
CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRINterval
```

class RfSettingsCls

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

get_eattenuation() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
value: float = driver.configure.gprf.measurement.rfSettings.get_eattenuation()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

return
rf_input_ext_att: No help available

get_envelope_power() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:ENPower
value: float = driver.configure.gprf.measurement.rfSettings.get_envelope_power()
```

Sets the expected nominal power of the measured RF signal.

return
exp_nominal_power: The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.

get_foffset() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FOFFset
value: float = driver.configure.gprf.measurement.rfSettings.get_foffset()
```

No command help available

return
freq_offset: No help available

get_frequency() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
value: float = driver.configure.gprf.measurement.rfSettings.get_frequency()
```

Selects the center frequency of the RF analyzer. For the supported frequency range, see 'Frequency ranges'.

return
analyzer_freq: No help available

get_lo_frequency() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LOFRequency
value: float = driver.configure.gprf.measurement.rfSettings.get_lo_frequency()
```

Queries the required external LO frequency resulting from the measurement settings. The command also triggers a refresh of the information before the query. So no need for a separate refresh command.

return
lo_frequency: No help available

get_lo_level() → LoLevel

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LOLevel
value: enums.LoLevel = driver.configure.gprf.measurement.rfSettings.get_lo_
↪level()
```

Queries whether the level of an external LO signal is correct.

return
lo_level: Level correct, too low, too high.

get_lr_interval() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRINterval
value: float = driver.configure.gprf.measurement.rfSettings.get_lr_interval()
```

Defines the measurement interval for level adjustment.

return
lvl_rang_interval: No help available

get_ml_offset() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:MLOFfset
value: float = driver.configure.gprf.measurement.rfSettings.get_ml_offset()
```

No command help available

return
mix_lev_offset: No help available

get_umargin() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:UMARgin
value: float = driver.configure.gprf.measurement.rfSettings.get_umargin()
```

Sets the margin that the measurement adds to the expected nominal power to determine the reference power. The reference power minus the external input attenuation must be within the power range of the selected input connector. Refer to the specifications document.

return
user_margin: No help available

set_eattenuation(*rf_input_ext_att: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
driver.configure.gprf.measurement.rfSettings.set_eattenuation(rf_input_ext_att␣
↪= 1.0)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

param rf_input_ext_att
No help available

set_envelope_power(*exp_nominal_power: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:ENPower
driver.configure.gprf.measurement.rfSettings.set_envelope_power(exp_nominal_
↪power = 1.0)
```

Sets the expected nominal power of the measured RF signal.

param exp_nominal_power
The range of the expected nominal power can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin
The input power range is stated in the specifications document.

set_foffset(*freq_offset: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FOFFset
driver.configure.gprf.measurement.rfSettings.set_foffset(freq_offset = 1.0)
```

No command help available

param freq_offset
No help available

set_frequency(*analyzer_freq: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
driver.configure.gprf.measurement.rfSettings.set_frequency(analyzer_freq = 1.0)
```

Selects the center frequency of the RF analyzer. For the supported frequency range, see 'Frequency ranges'.

param analyzer_freq
No help available

set_lr_interval(*lvl_rang_interval: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRINterval
driver.configure.gprf.measurement.rfSettings.set_lr_interval(lvl_rang_interval␣
↪= 1.0)
```

Defines the measurement interval for level adjustment.

param lvl_rang_interval
No help available

set_ml_offset(*mix_lev_offset: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:MLOFfset
driver.configure.gprf.measurement.rfSettings.set_ml_offset(mix_lev_offset = 1.0)
```

No command help available

param mix_lev_offset
No help available

set_umargin(*user_margin: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:UMARgin
driver.configure.gprf.measurement.rfSettings.set_umargin(user_margin = 1.0)
```

Sets the margin that the measurement adds to the expected nominal power to determine the reference power. The reference power minus the external input attenuation must be within the power range of the selected input connector. Refer to the specifications document.

param user_margin
No help available

Cloning the Group

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

Subgroups

6.4.1.2.10.1 LrStart

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRSTart
```

class LrStartCls

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

set() → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRSTart
driver.configure.gprf.measurement.rfSettings.lrStart.set()
```

Starts level adjustment.

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRSTart
driver.configure.gprf.measurement.rfSettings.lrStart.set_with_opc()
```

Starts level adjustment.

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

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

6.4.1.2.11 Scenario

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:SCENario[:ACTivate]
```

class ScenarioCls

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

get_activate() → MeasScenario

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SCENario[:ACTivate]
value: enums.MeasScenario = driver.configure.gprf.measurement.scenario.get_
↪activate()
```

No command help available

return
scenario: No help available

set_activate(*scenario: MeasScenario*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SCENario[:ACTivate]
driver.configure.gprf.measurement.scenario.set_activate(scenario = enums.
↪MeasScenario.CSPath)
```

No command help available

param scenario
No help available

6.4.1.2.12 Spectrum

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
```

class SpectrumCls

Spectrum commands group definition. 24 total commands, 3 Subgroups, 4 group commands

get_amode() → AveragingMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
value: enums.AveragingMode = driver.configure.gprf.measurement.spectrum.get_
↪amode()
```

No command help available

return
averaging_mode: No help available

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
value: enums.Repeat = driver.configure.gprf.measurement.spectrum.get_
↪repetition()
```

No command help available

return
repetition: No help available

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
value: int = driver.configure.gprf.measurement.spectrum.get_scount()
```

No command help available

return
statistic_count: No help available

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
value: float = driver.configure.gprf.measurement.spectrum.get_timeout()
```

No command help available

return
tcd_timeout: No help available

set_amode(*averaging_mode: AveragingMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
driver.configure.gprf.measurement.spectrum.set_amode(averaging_mode = enums.
↪AveragingMode.LINear)
```

No command help available

param averaging_mode
No help available

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
driver.configure.gprf.measurement.spectrum.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

No command help available

param repetition
No help available

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
driver.configure.gprf.measurement.spectrum.set_scount(statistic_count = 1)
```

No command help available

param statistic_count
No help available

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
driver.configure.gprf.measurement.spectrum.set_timeout(tcd_timeout = 1.0)
```

No command help available

param tcd_timeout
No help available

Cloning the Group

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

Subgroups

6.4.1.2.12.1 FreqSweep

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:DEBug
```

class FreqSweepCls

FreqSweep commands group definition. 7 total commands, 3 Subgroups, 1 group commands

class DebugStruct

Structure for reading output parameters. Fields:

- Span: float: No parameter help available
- Rbw_Auto: bool: No parameter help available
- Rbw: float: No parameter help available
- Vbw_Auto: bool: No parameter help available
- Vbw: float: No parameter help available
- Swt_Auto: bool: No parameter help available
- Swt: float: No parameter help available
- Kfactor: float: No parameter help available
- Rbw_Index: int: No parameter help available
- Vbw_Index: int: No parameter help available
- Vbw_Enable: bool: No parameter help available
- Error_Reason: int: No parameter help available

get_debug() → DebugStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:DEBug
value: DebugStruct = driver.configure.gprf.measurement.spectrum.freqSweep.get_
↪debug()
```

No command help available

return

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

Cloning the Group

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

Subgroups

6.4.1.2.12.2 Rbw

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW:AUTO
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW
```

class RbwCls

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

get_auto() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW:AUTO
value: bool = driver.configure.gprf.measurement.spectrum.freqSweep.rbw.get_
↪auto()
```

No command help available

return
rbw_auto: No help available

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW
value: float = driver.configure.gprf.measurement.spectrum.freqSweep.rbw.get_
↪value()
```

No command help available

return
rbw: No help available

set_auto(*rbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW:AUTO
driver.configure.gprf.measurement.spectrum.freqSweep.rbw.set_auto(rbw_auto =␣
↪False)
```

No command help available

param rbw_auto
No help available

set_value(*rbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW
driver.configure.gprf.measurement.spectrum.freqSweep.rbw.set_value(rbw = 1.0)
```

No command help available

param rbw
No help available

6.4.1.2.12.3 Swt

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT:AUTO
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT
```

class SwtCls

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

get_auto() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT:AUTO
value: bool = driver.configure.gprf.measurement.spectrum.freqSweep.swt.get_
↪auto()
```

No command help available

return
sweep_time_auto: No help available

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT
value: float = driver.configure.gprf.measurement.spectrum.freqSweep.swt.get_
↪value()
```

No command help available

return
sweep_time: No help available

set_auto(*sweep_time_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT:AUTO
driver.configure.gprf.measurement.spectrum.freqSweep.swt.set_auto(sweep_time_
↪auto = False)
```

No command help available

param sweep_time_auto
No help available

set_value(*sweep_time: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT
driver.configure.gprf.measurement.spectrum.freqSweep.swt.set_value(sweep_time =␣
↪1.0)
```

No command help available

param sweep_time
No help available

6.4.1.2.12.4 Vbw

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW:AUTO
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW
```

class VbwCls

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

get_auto() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW:AUTO
value: bool = driver.configure.gprf.measurement.spectrum.freqSweep.vbw.get_
↪auto()
```

No command help available

return
vbw_auto: No help available

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW
value: float or bool = driver.configure.gprf.measurement.spectrum.freqSweep.vbw.
↪get_value()
```

No command help available

return
vbw: (float or boolean) No help available

set_auto(*vbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW:AUTO
driver.configure.gprf.measurement.spectrum.freqSweep.vbw.set_auto(vbw_auto =␣
↪False)
```

No command help available

param vbw_auto
No help available

set_value(*vbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW
driver.configure.gprf.measurement.spectrum.freqSweep.vbw.set_value(vbw = 1.0)
```

No command help available

param vbw
(float or boolean) No help available

6.4.1.2.12.5 Frequency

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:CENTer
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STARt
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:LASPan
```

class FrequencyCls

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

get_center() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:CENTer
value: float = driver.configure.gprf.measurement.spectrum.frequency.get_center()
```

No command help available

return
center_frequency: No help available

get_laspan() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:LASPan
value: float = driver.configure.gprf.measurement.spectrum.frequency.get_laspan()
```

No command help available

return
last_span: No help available

get_start() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STARt
value: float = driver.configure.gprf.measurement.spectrum.frequency.get_start()
```

No command help available

return
start_frequency: No help available

get_stop() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
value: float = driver.configure.gprf.measurement.spectrum.frequency.get_stop()
```

No command help available

return
stop_frequency: No help available

set_center(*center_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:CENTer
driver.configure.gprf.measurement.spectrum.frequency.set_center(center_
↪frequency = 1.0)
```

No command help available

param center_frequency
No help available

set_laspan(*last_span: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:LASPan
driver.configure.gprf.measurement.spectrum.frequency.set_laspan(last_span = 1.0)
```

No command help available

param last_span
No help available

set_start(*start_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STARt
driver.configure.gprf.measurement.spectrum.frequency.set_start(start_frequency␣
↪= 1.0)
```

No command help available

param start_frequency
No help available

set_stop(*stop_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
driver.configure.gprf.measurement.spectrum.frequency.set_stop(stop_frequency =␣
↪1.0)
```

No command help available

param stop_frequency
No help available

Cloning the Group

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

Subgroups

6.4.1.2.12.6 Span

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN:MODE
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN
```

class SpanCls

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

get_mode() → SpanMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN:MODE
value: enums.SpanMode = driver.configure.gprf.measurement.spectrum.frequency.
↪span.get_mode()
```

No command help available

return
span_mode: No help available

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN
value: float = driver.configure.gprf.measurement.spectrum.frequency.span.get_
↪value()
```

No command help available

return
frequency_span: No help available

set_mode(*span_mode: SpanMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN:MODE
driver.configure.gprf.measurement.spectrum.frequency.span.set_mode(span_mode =␣
↪enums.SpanMode.FSWeep)
```

No command help available

param span_mode
No help available

set_value(*frequency_span: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN
driver.configure.gprf.measurement.spectrum.frequency.span.set_value(frequency_
↪span = 1.0)
```

No command help available

param frequency_span
No help available

6.4.1.2.12.7 ZeroSpan

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:DEBug
```

class ZeroSpanCls

ZeroSpan commands group definition. 7 total commands, 2 Subgroups, 2 group commands

class DebugStruct

Structure for reading output parameters. Fields:

- Rbw: float: No parameter help available
- Vbw_Auto: bool: No parameter help available
- Vbw: float: No parameter help available
- Swt: float: No parameter help available
- Rbw_Index: int: No parameter help available
- Vbw_Index: int: No parameter help available
- Vbw_Enable: bool: No parameter help available
- Error_Reason: int: No parameter help available

get_debug() → DebugStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:DEBug
value: DebugStruct = driver.configure.gprf.measurement.spectrum.zeroSpan.get_
↪debug()
```

No command help available

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

get_swt() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
value: float = driver.configure.gprf.measurement.spectrum.zeroSpan.get_swt()
```

No command help available

return
sweep_time: No help available

set_swt(*sweep_time: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
driver.configure.gprf.measurement.spectrum.zeroSpan.set_swt(sweep_time = 1.0)
```

No command help available

param sweep_time
No help available

Cloning the Group

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

Subgroups

6.4.1.2.12.8 Rbw

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:TYPE
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:BANDpass
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:GAUSs
```

class RbwCls

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

get_bandpass() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:BANDpass
value: float = driver.configure.gprf.measurement.spectrum.zeroSpan.rbw.get_
↪bandpass()
```

No command help available

return
rbw_bandpass: No help available

get_gauss() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:GAUSs
value: float = driver.configure.gprf.measurement.spectrum.zeroSpan.rbw.get_
↪gauss()
```

No command help available

return
rbw: No help available

get_type_py() → RbwFilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:TYPE
value: enums.RbwFilterType = driver.configure.gprf.measurement.spectrum.
↪zeroSpan.rbw.get_type_py()
```

No command help available

return
rbw_type: No help available

set_bandpass(*rbw_bandpass: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:BANDpass
driver.configure.gprf.measurement.spectrum.zeroSpan.rbw.set_bandpass(rbw_
↪bandpass = 1.0)
```

No command help available

param rbw_bandpass
No help available

set_gauss(*rbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:GAUSs
driver.configure.gprf.measurement.spectrum.zeroSpan.rbw.set_gauss(rbw = 1.0)
```

No command help available

param rbw
No help available

set_type_py(*rbw_type: RbwFilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:TYPE
driver.configure.gprf.measurement.spectrum.zeroSpan.rbw.set_type_py(rbw_type =␣
↪enums.RbwFilterType.BANDpass)
```

No command help available

param rbw_type
No help available

6.4.1.2.12.9 Vbw

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW:AUTO
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW
```

class VbwCls

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

get_auto() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW:AUTO
value: bool = driver.configure.gprf.measurement.spectrum.zeroSpan.vbw.get_auto()
```

No command help available

return
vbw_auto: No help available

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW
value: float or bool = driver.configure.gprf.measurement.spectrum.zeroSpan.vbw.
↪get_value()
```

No command help available

return
vbw: (float or boolean) No help available

set_auto(*vbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW:AUTO
driver.configure.gprf.measurement.spectrum.zeroSpan.vbw.set_auto(vbw_auto =␣
↪False)
```

No command help available

param vbw_auto
No help available

set_value(*vbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW
driver.configure.gprf.measurement.spectrum.zeroSpan.vbw.set_value(vbw = 1.0)
```

No command help available

param vbw
(float or boolean) No help available

6.4.2 System

class SystemCls

System commands group definition. 21 total commands, 12 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.2.1 Attenuation

class AttenuationCls

Attenuation 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.system.attenuation.clone()
```

Subgroups

6.4.2.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.system.attenuation.correctionTable.clone()
```

Subgroups

6.4.2.1.1.1 Info

class InfoCls

Info 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.system.attenuation.correctionTable.info.clone()
```

Subgroups

6.4.2.1.1.2 Globale

SCPI Command :

```
[CONFigure]:SYSTem:ATTenuation:CTABle:INFO:GLOBal
```

class GlobaleCls

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

class GetStruct

Response structure. Fields:

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

get(*name: str*) → GetStruct

```
# SCPI: [CONFigure]:SYSTem:ATTenuation:CTABle:INFO:GLOBal
value: GetStruct = driver.configure.system.attenuation.correctionTable.info.
↪globale.get(name = 'abc')
```

No command help available

param name
No help available

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

6.4.2.1.1.3 Tenvironment

SCPI Command :

```
[CONFigure]:SYSTem:ATTenuation:CTABle:INFO[:TENVironment]
```

`class TenvironmentCls`

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

`class GetStruct`

Response structure. Fields:

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

`get`(*name: str*) → GetStruct

```
# SCPI: [CONFigure]:SYSTem:ATTenuation:CTABle:INFO[:TENVironment]
value: GetStruct = driver.configure.system.attenuation.correctionTable.info.
↪tenvironment.get(name = 'abc')
```

No command help available

param name
No help available

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

6.4.2.2 Control

`class ControlCls`

Control 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.configure.system.control.clone()
```

Subgroups

6.4.2.2.1 Reboot

SCPI Command :

```
[CONFigure]:SYSTem:CONTrol:REBoot
```

class RebootCls

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

set() → None

```
# SCPI: [CONFigure]:SYSTem:CONTrol:REBoot
driver.configure.system.control.reboot.set()
```

No command help available

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

```
# SCPI: [CONFigure]:SYSTem:CONTrol:REBoot
driver.configure.system.control.reboot.set_with_opc()
```

No command help available

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

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

6.4.2.2.2 Restart

SCPI Command :

```
[CONFigure]:SYSTem:CONTrol:RESTart
```

class RestartCls

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

set() → None

```
# SCPI: [CONFigure]:SYSTem:CONTrol:RESTart
driver.configure.system.control.restart.set()
```

No command help available

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

```
# SCPI: [CONFigure]:SYSTem:CONTrol:RESTart
driver.configure.system.control.restart.set_with_opc()
```

No command help available

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

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

6.4.2.2.3 Shutdown

SCPI Command :

```
[CONFigure]:SYSTem:CONTrol:SHUTdown
```

class ShutdownCls

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

set() → None

```
# SCPI: [CONFigure]:SYSTem:CONTrol:SHUTdown
driver.configure.system.control.shutdown.set()
```

No command help available

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

```
# SCPI: [CONFigure]:SYSTem:CONTrol:SHUTdown
driver.configure.system.control.shutdown.set_with_opc()
```

No command help available

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

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

6.4.2.3 Edevice

SCPI Command :

```
[CONFigure]:SYSTem:EDEVice
```

class EdeviceCls

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

class EdeviceStruct

Response structure. Fields:

- Device_Type: enums.DeviceType: No parameter help available
- Device_Mode: enums.DeviceMode: No parameter help available

get() → EdeviceStruct

```
# SCPI: [CONFigure]:SYSTem:EDEVice
value: EdeviceStruct = driver.configure.system.edevice.get()
```

No command help available

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

set(*device_type: DeviceType*, *device_mode: DeviceMode*) → None

```
# SCPI: [CONFigure]:SYSTem:EDEVice
driver.configure.system.edevice.set(device_type = enums.DeviceType.NONE, device_
↪mode = enums.DeviceMode.M2X2)
```

No command help available

param device_type
No help available

param device_mode
No help available

6.4.2.4 Positioner<Positioner>

RepCap Settings

```
# Range: Ix1 .. Ix32
rc = driver.configure.system.positioner.repcap_positioner_get()
driver.configure.system.positioner.repcap_positioner_set(repcap.Positioner.Ix1)
```

class PositionerCls

Positioner commands group definition. 4 total commands, 4 Subgroups, 0 group commands Repeated Capability: Positioner, default value after init: Positioner.Ix1

Cloning the Group

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

Subgroups

6.4.2.4.1 HwProperties

SCPI Command :

```
[CONFigure]:SYSTem:POSitioner<PositionerIdx>:HWPRoperties
```

class HwPropertiesCls

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

class GetStruct

Response structure. Fields:

- Key: str: No parameter help available
- Value: str: No parameter help available

get(*positioner=Positioner.Default*) → GetStruct

```
# SCPI: [CONFigure]:SYSTem:POSitioner<PositionerIdx>:HWPRoperties
value: GetStruct = driver.configure.system.positioner.hwProperties.
↪get(positioner = repcap.Positioner.Default)
```

No command help available

param positioner
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Positioner')

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

6.4.2.4.2 Move

class MoveCls

Move 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.system.positioner.move.clone()
```

Subgroups

6.4.2.4.2.1 To

SCPI Command :

```
[CONFigure]:SYSTem:POSitioner<PositionerIdx>:MOVE:TO
```

class ToCls

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

set(*theta: float*, *phi: float*, *positioner=Positioner.Default*) → None

```
# SCPI: [CONFigure]:SYSTem:POSitioner<PositionerIdx>:MOVE:TO
driver.configure.system.positioner.move.to.set(theta = 1.0, phi = 1.0,␣
↪positioner = repcap.Positioner.Default)
```

No command help available

param theta
No help available

param phi
No help available

param positioner
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Positioner')

6.4.2.4.3 Moving

SCPI Command :

```
[CONFigure]:SYSTem:POSitioner<PositionerIdx>:MOVing:STOP
```

class MovingCls

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

stop(*positioner=Positioner.Default*) → None

```
# SCPI: [CONFigure]:SYSTem:POSitioner<PositionerIdx>:MOVing:STOP
driver.configure.system.positioner.moving.stop(positioner = repcap.Positioner.
↪Default)
```

No command help available

param positioner
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Positioner')

stop_with_opc(*positioner=Positioner.Default*, *opc_timeout_ms: int = -1*) → None

6.4.2.4.4 Versions

SCPI Command :

```
[CONFigure]:SYSTem:POSitioner<PositionerIdx>:VERSions
```

class VersionsCls

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

class GetStruct

Response structure. Fields:

- Firmware_Vers: str: No parameter help available
- Driver_Vers: str: No parameter help available

get(*positioner=Positioner.Default*) → GetStruct

```
# SCPI: [CONFigure]:SYSTem:POSitioner<PositionerIdx>:VERSions
value: GetStruct = driver.configure.system.positioner.versions.get(positioner =␣
↪repcap.Positioner.Default)
```

No command help available

param positioner
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Positioner')

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

6.4.2.5 Recall

class RecallCls

Recall 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.system.recall.clone()
```

Subgroups

6.4.2.5.1 Partial

SCPI Command :

```
[CONFigure]:SYSTem:RECall:PARTial
```

class PartialCls

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

get(*saving_path: str*) → List[str]

```
# SCPI: [CONFigure]:SYSTem:RECall:PARTial
value: List[str] = driver.configure.system.recall.partial.get(saving_path = 'abc
↪')
```

No command help available

param saving_path
No help available

return
saving_module: No help available

set(*saving_path: str*, *saving_module: List[str] = None*) → None

```
# SCPI: [CONFigure]:SYSTem:RECall:PARTial
driver.configure.system.recall.partial.set(saving_path = 'abc', saving_module =␣
↪['abc1', 'abc2', 'abc3'])
```

No command help available

param saving_path
No help available

param saving_module
No help available

6.4.2.6 Reset

SCPI Command :

```
[CONFigure]:SYSTem:RESet:PARTial
```

class ResetCls

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

set_partial(*resetable_system_part: List[str]*) → None

```
# SCPI: [CONFigure]:SYSTem:RESet:PARTial
driver.configure.system.reset.set_partial(resetable_system_part = ['abc1', 'abc2
↪', 'abc3'])
```

No command help available

param resetable_system_part
No help available

6.4.2.7 Rf42

class Rf42Cls

Rf42 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.system.rf42.clone()
```

Subgroups

6.4.2.7.1 Box<Box>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.configure.system.rf42.box.repcap_box_get()
driver.configure.system.rf42.box.repcap_box_set(repcap.Box.Nr1)
```

class BoxCls

Box commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: Box, default value after init: Box.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.system.rf42.box.clone()
```

Subgroups

6.4.2.7.1.1 Apreset

class ApresetCls

Apreset 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.system.rf42.box.apreset.clone()
```

Subgroups

6.4.2.7.1.2 Rx

SCPI Command :

```
[CONFigure]:SYSTem:RF42:BOX<BoxNo>:APReset:RX
```

class RxCls

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

get(*box=Box.Default*) → Amplification

```
# SCPI: [CONFigure]:SYSTem:RF42:BOX<BoxNo>:APReset:RX
value: enums.Amplification = driver.configure.system.rf42.box.apreset.rx.
↪get(box = repcap.Box.Default)
```

No command help available

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

return
amplification: No help available

set(*amplification: Amplification*, *box=Box.Default*) → None

```
# SCPI: [CONFigure]:SYSTem:RF42:BOX<BoxNo>:APReset:RX
driver.configure.system.rf42.box.apreset.rx.set(amplification = enums.
↪Amplification.HIGH, box = repcap.Box.Default)
```

No command help available

param amplification
No help available

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

6.4.2.7.1.3 Tx

SCPI Command :

```
[CONFigure]:SYSTem:RF42:BOX<BoxNo>:APReset:TX
```

class TxCls

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

get(*box=Box.Default*) → LowHigh

```
# SCPI: [CONFigure]:SYSTem:RF42:BOX<BoxNo>:APReset:TX
value: enums.LowHigh = driver.configure.system.rf42.box.apreset.tx.get(box =␣
↪repcap.Box.Default)
```

No command help available

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

return
amplification: No help available

set(*amplification: LowHigh*, *box=Box.Default*) → None

```
# SCPI: [CONFigure]:SYSTem:RF42:BOX<BoxNo>:APReset:TX
driver.configure.system.rf42.box.apreset.tx.set(amplification = enums.LowHigh.
↪HIGH, box = repcap.Box.Default)
```

No command help available

param amplification
No help available

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

6.4.2.8 Rrhead

class RrheadCls

Rrhead 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.system.rrhead.clone()
```

Subgroups

6.4.2.8.1 Lo

class LoCls

Lo 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.system.rrhead.lo.clone()
```

Subgroups

6.4.2.8.1.1 Source

class SourceCls

Source 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.system.rrhead.lo.source.clone()
```

Subgroups

6.4.2.8.1.2 Rx

SCPI Command :

```
CONFigure:SYSTem:RRHead:LO:SOURce:RX
```

class RxCls

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

get(*rrh_name: str*) → SourceInt

```
# SCPI: CONFigure:SYSTem:RRHead:LO:SOURce:RX
value: enums.SourceInt = driver.configure.system.rrhead.lo.source.rx.get(rrh_
↪name = 'abc')
```

No command help available

param rrh_name
No help available

return
source: No help available

set(*rrh_name: str*, *source: SourceInt*) → None

```
# SCPI: CONFigure:SYSTem:RRHead:LO:SOURce:RX
driver.configure.system.rrhead.lo.source.rx.set(rrh_name = 'abc', source =␣
↪enums.SourceInt.EXTernal)
```

No command help available

param rrh_name
No help available

param source
No help available

6.4.2.8.1.3 Tx

SCPI Command :

```
CONFigure:SYSTem:RRHead:LO:SOURce:TX
```

class TxCls

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

get(*rrh_name: str*) → SourceInt

```
# SCPI: CONFigure:SYSTem:RRHead:LO:SOURce:TX
value: enums.SourceInt = driver.configure.system.rrhead.lo.source.tx.get(rrh_
↪name = 'abc')
```

No command help available

param rrh_name
No help available

return
source: No help available

set(*rrh_name: str*, *source: SourceInt*) → None

```
# SCPI: CONFigure:SYSTem:RRHead:LO:SOURce:TX
driver.configure.system.rrhead.lo.source.tx.set(rrh_name = 'abc', source =␣
↪enums.SourceInt.EXTernal)
```

No command help available

param rrh_name
No help available

param source
No help available

6.4.2.9 Save

class SaveCls

Save 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.system.save.clone()
```

Subgroups

6.4.2.9.1 Partial

SCPI Command :

```
[CONFigure]:SYSTem:SAVE:PARTial
```

class PartialCls

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

set(*saving_path: str*, *saving_module: List[str]*) → None

```
# SCPI: [CONFigure]:SYSTem:SAVE:PARTial
driver.configure.system.save.partial.set(saving_path = 'abc', saving_module = [
↪'abc1', 'abc2', 'abc3'])
```

No command help available

param saving_path
No help available

param saving_module
No help available

6.4.2.10 Vse

SCPI Commands :

```
[CONFigure]:SYSTem:VSE:CONNect
[CONFigure]:SYSTem:VSE:DISConnect
```

class VseCls

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

get_connect() → str

```
# SCPI: [CONFigure]:SYSTem:VSE:CONNect
value: str = driver.configure.system.vse.get_connect()
```

No command help available

return
address: No help available

get_disconnect() → str

```
# SCPI: [CONFigure]:SYSTem:VSE:DISConnect
value: str = driver.configure.system.vse.get_disconnect()
```

No command help available

return
address: No help available

set_connect(*address: str*) → None

```
# SCPI: [CONFigure]:SYSTem:VSE:CONNect
driver.configure.system.vse.set_connect(address = 'abc')
```

No command help available

param address
No help available

set_disconnect(*address: str*) → None

```
# SCPI: [CONFigure]:SYSTem:VSE:DISConnect
driver.configure.system.vse.set_disconnect(address = 'abc')
```

No command help available

param address
No help available

6.4.2.11 Z310

class Z310Cls

Z310 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.system.z310.clone()
```

Subgroups

6.4.2.11.1 Attenuation

SCPI Command :

```
[CONFigure]:SYSTem:Z310:ATTenuation
```

class AttenuationCls

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

get(*connector_name: str*) → float

```
# SCPI: [CONFigure]:SYSTem:Z310:ATTenuation
value: float = driver.configure.system.z310.attenuation.get(connector_name =
↪'abc')
```

No command help available

param connector_name
No help available

return
attenuation: No help available

set(*attenuation: float*) → None

```
# SCPI: [CONFigure]:SYSTem:Z310:ATTenuation
driver.configure.system.z310.attenuation.set(attenuation = 1.0)
```

No command help available

param attenuation
No help available

6.4.2.12 Z320

class Z320Cls

Z320 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.system.z320.clone()
```

Subgroups

6.4.2.12.1 Attenuation

SCPI Command :

```
[CONFigure]:SYSTem:Z320:ATTenuation
```

class AttenuationCls

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

get(*connector_name: str*) → float

```
# SCPI: [CONFigure]:SYSTem:Z320:ATTenuation
value: float = driver.configure.system.z320.attenuation.get(connector_name =
↪'abc')
```

No command help available

param connector_name
No help available

return
attenuation: No help available

set(*attenuation: float*) → None

```
# SCPI: [CONFigure]:SYSTem:Z320:ATTenuation
driver.configure.system.z320.attenuation.set(attenuation = 1.0)
```

No command help available

param attenuation
No help available

6.4.3 Tenvironment

class TenvironmentCls

Tenvironment commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.3.1 Spath

class SpathCls

Spath commands group definition. 6 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.3.1.1 Attenuation

class AttenuationCls

Attenuation 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.tenvironment.spath.attenuation.clone()
```

Subgroups

6.4.3.1.1.1 Rx

SCPI Command :

```
[CONFigure]:TENVironment:SPATh:ATTenuation:RX
```

class RxCls

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

get(*name_signal_path: str*) → float

```
# SCPI: [CONFigure]:TENVironment:SPATh:ATTenuation:RX
value: float = driver.configure.tenvironment.spath.attenuation.rx.get(name_
↪signal_path = 'abc')
```

No command help available

param name_signal_path
No help available

return
value: No help available

set(*name_signal_path: str*, *value: float*) → None

```
# SCPI: [CONFigure]:TENVironment:SPATh:ATTenuation:RX
driver.configure.tenvironment.spath.attenuation.rx.set(name_signal_path = 'abc',
↪ value = 1.0)
```

No command help available

param name_signal_path
No help available

param value
No help available

6.4.3.1.1.2 Tx

SCPI Command :

```
[CONFigure]:TENVironment:SPATh:ATTenuation:TX
```

class TxCls

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

get(*name_signal_path: str*) → float

```
# SCPI: [CONFigure]:TENVironment:SPATh:ATTenuation:TX
value: float = driver.configure.tenvironment.spath.attenuation.tx.get(name_
↪signal_path = 'abc')
```

No command help available

param name_signal_path
No help available

return
value: No help available

set(*name_signal_path: str*, *value: float*) → None

```
# SCPI: [CONFigure]:TENVironment:SPATh:ATTenuation:TX
driver.configure.tenvironment.spath.attenuation.tx.set(name_signal_path = 'abc',
↪ value = 1.0)
```

No command help available

param name_signal_path
No help available

param value
No help available

6.4.3.1.2 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.tenvironment.spath.correctionTable.clone()
```

Subgroups

6.4.3.1.2.1 Rx

SCPI Command :

```
[CONFigure]:TENVironment:SPATh:CTABle:RX
```

class RxCls

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

get(*name_signal_path: str*) → List[str]

```
# SCPI: [CONFigure]:TENVironment:SPATh:CTABle:RX
value: List[str] = driver.configure.tenvironment.spath.correctionTable.rx.
↪get(name_signal_path = 'abc')
```

No command help available

param name_signal_path
No help available

return
correction_table: No help available

set(*name_signal_path: str*, *correction_table: List[str]*) → None

```
# SCPI: [CONFigure]:TENVironment:SPATh:CTABle:RX
driver.configure.tenvironment.spath.correctionTable.rx.set(name_signal_path =
↪'abc', correction_table = ['abc1', 'abc2', 'abc3'])
```

No command help available

param name_signal_path
No help available

param correction_table
No help available

6.4.3.1.2.2 Tx

SCPI Command :

```
[CONFigure]:TENVironment:SPATh:CTABle:TX
```

class TxCls

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

get(*name_signal_path: str*) → List[str]

```
# SCPI: [CONFigure]:TENVironment:SPATh:CTABle:TX
value: List[str] = driver.configure.tenvironment.spath.correctionTable.tx.
↪get(name_signal_path = 'abc')
```

No command help available

param name_signal_path
No help available

return
correction_table: No help available

set(*name_signal_path: str*, *correction_table: List[str]*) → None

```
# SCPI: [CONFigure]:TENVironment:SPATh:CTABle:TX
driver.configure.tenvironment.spath.correctionTable.tx.set(name_signal_path =
↪'abc', correction_table = ['abc1', 'abc2', 'abc3'])
```

No command help available

param name_signal_path
No help available

param correction_table
No help available

6.4.3.1.3 Direction

SCPI Command :

```
[CONFigure]:TENVironment:SPATh:DIRection
```

class DirectionCls

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

get(*name_signal_path: str*) → SignalDirection

```
# SCPI: [CONFigure]:TENVironment:SPATh:DIRection
value: enums.SignalDirection = driver.configure.tenvironment.spath.direction.
↪get(name_signal_path = 'abc')
```

No command help available

param name_signal_path
No help available

return
signal_direction: No help available

`set`(*name_signal_path: str*, *signal_direction: SignalDirection*) → None

```
# SCPI: [CONFigure]:TENVironment:SPATh:DIRection
driver.configure.tenvironment.spath.direction.set(name_signal_path = 'abc',␣
↪signal_direction = enums.SignalDirection.RX)
```

No command help available

param name_signal_path
No help available

param signal_direction
No help available

6.4.3.1.4 Info

SCPI Command :

```
[CONFigure]:TENVironment:SPATh:INFO
```

`class InfoCls`

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

`class GetStruct`

Response structure. Fields:

- Name_Antenna: str: No parameter help available
- Name_Connector: str: No parameter help available
- Signal_Direction: enums.SignalDirection: No parameter help available
- No_Corr_Table_Rx: float: No parameter help available
- Corr_Table_Rx: str: No parameter help available
- No_Corr_Table_Tx: float: No parameter help available
- Corr_Table_Tx: List[str]: No parameter help available

`get`(*name_spath: str*) → GetStruct

```
# SCPI: [CONFigure]:TENVironment:SPATh:INFO
value: GetStruct = driver.configure.tenvironment.spath.info.get(name_spath =
↪'abc')
```

No command help available

param name_spath
No help available

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

6.5 Create

class CreateCls

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

Cloning the Group

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

Subgroups

6.5.1 System

class SystemCls

System 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.create.system.clone()
```

Subgroups

6.5.1.1 Attenuation

class AttenuationCls

Attenuation 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.create.system.attenuation.clone()
```

Subgroups

6.5.1.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.create.system.attenuation.correctionTable.clone()
```

Subgroups

6.5.1.1.1.1 Globale

SCPI Command :

```
CREate:SYSTem:ATTenuation:CTABle:GLOBal
```

class GlobaleCls

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

set(*name: str*, *arg_1: List[float] = None*, *attenuation: List[float] = None*) → None

```
# SCPI: CREate:SYSTem:ATTenuation:CTABle:GLOBal
driver.create.system.attenuation.correctionTable.globale.set(name = 'abc', arg_
↪1 = [1.1, 2.2, 3.3], attenuation = [1.1, 2.2, 3.3])
```

No command help available

param name
No help available

param arg_1
No help available

param attenuation
No help available

6.5.1.1.1.2 Tenvironment

SCPI Command :

```
CREate:SYSTem:ATTenuation:CTABle[:TENVironment]
```

class TenvironmentCls

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

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

```
# SCPI: CREate:SYSTem:ATTenuation:CTABle[:TENVironment]
driver.create.system.attenuation.correctionTable.tenvironment.set(name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3], attenuation = [1.1, 2.2, 3.3])
```

No command help available

param name
No help available

param frequency
No help available

param attenuation
No help available

6.5.2 Tenvironment

class TenvironmentCls

Tenvironment 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.create.tenvironment.clone()
```

Subgroups

6.5.2.1 Spath

SCPI Command :

```
CREate:TENVironment:SPATh
```

class SpathCls

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

set(*name_signal_path: str*, *name_antenna: str*, *name_connector: str*, *overwrite: bool = None*) → None

```
# SCPI: CREate:TENVironment:SPATh
driver.create.tenvironment.spath.set(name_signal_path = 'abc', name_antenna =
↪'abc', name_connector = 'abc', overwrite = False)
```

No command help available

param name_signal_path
No help available

param name_antenna
No help available

param name_connector
No help available

param overwrite
No help available

6.6 Diagnostic

class DiagnosticCls

Diagnostic commands group definition. 58 total commands, 8 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.1 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.diagnostic.catalog.clone()
```

Subgroups

6.6.1.1 System

class SystemCls

System 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.catalog.system.clone()
```

Subgroups

6.6.1.1.1 Connectors

SCPI Command :

```
DIAGnostic:CATalog:SYSTem:CONNectors
```

class ConnectorsCls

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

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

```
# SCPI: DIAGnostic:CATalog:SYSTem:CONNectors
value: List[str] = driver.diagnostic.catalog.system.connectors.get(name_style =␣
↪enums.NameStyle.FQName)
```

No command help available

param name_style
No help available

return
name_connector: No help available

6.6.2 Configure

class ConfigureCls

Configure commands group definition. 19 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.2.1 Gprf

class GprfCls

Gprf 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.configure.gprf.clone()
```

Subgroups

6.6.2.1.1 Measurement

SCPI Command :

```
DIAGnostic:CONFigure:GPRF:MEASurement:LOGGing
```

class MeasurementCls

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

get_logging() → bool

```
# SCPI: DIAGnostic:CONFigure:GPRF:MEASurement:LOGGing
value: bool = driver.diagnostic.configure.gprf.measurement.get_logging()
```

No command help available

return
logging: No help available

set_logging(*logging: bool*) → None

```
# SCPI: DIAGnostic:CONFigure:GPRF:MEASurement:LOGGing
driver.diagnostic.configure.gprf.measurement.set_logging(logging = False)
```

No command help available

param logging
No help available

6.6.2.2 System

class SystemCls

System commands group definition. 18 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.2.2.1 Dapi

class DapiCls

Dapi commands group definition. 16 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.2.2.1.1 Logging

class LoggingCls

Logging commands group definition. 16 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.2.2.1.2 File

class FileCls

File 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.configure.system.dapi.logging.file.clone()
```

Subgroups

6.6.2.2.1.3 Psub

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:PAYLoad
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB
```

class PsubCls

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

get_payload() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:PAYLoad
value: bool = driver.diagnostic.configure.system.dapi.logging.file.psub.get_
↪payload()
```

No command help available

return
payload: No help available

get_value() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB
value: bool = driver.diagnostic.configure.system.dapi.logging.file.psub.get_
↪value()
```

No command help available

return
enable: No help available

set_payload(*payload: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:PAYLoad
driver.diagnostic.configure.system.dapi.logging.file.psub.set_payload(payload =␣
↪False)
```

No command help available

param payload
No help available

set_value(*enable: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB
driver.diagnostic.configure.system.dapi.logging.file.psub.set_value(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.configure.system.dapi.logging.file.psub.clone()
```

Subgroups

6.6.2.2.1.4 FilterPy

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:FILTer:MNAMe
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:FILTer:RNAMe
```

class FilterPyCls

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

get_mname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:FILTer:MNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.file.psub.filterPy.
↪get_mname()
```

No command help available

return
filter_mname: No help available

get_rname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:FILTer:RNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.file.psub.filterPy.
↪get_rname()
```

No command help available

return
filter_rname: No help available

set_mname(*filter_mname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:FILTer:MNAMe
driver.diagnostic.configure.system.dapi.logging.file.psub.filterPy.set_
↪mname(filter_mname = 'abc')
```

No command help available

param filter_mname
No help available

set_rname(*filter_rname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:PSUB:FILTer:RNAMe
driver.diagnostic.configure.system.dapi.logging.file.psub.filterPy.set_
↪rname(filter_rname = 'abc')
```

No command help available

param filter_rname
No help available

6.6.2.2.1.5 Rpc

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:PAYLoad
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC
```

class RpcCls

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

get_payload() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:PAYLoad
value: bool = driver.diagnostic.configure.system.dapi.logging.file.rpc.get_
↪payload()
```

No command help available

return
payload: No help available

get_value() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC
value: bool = driver.diagnostic.configure.system.dapi.logging.file.rpc.get_
↪value()
```

No command help available

return
enable: No help available

set_payload(*payload: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:PAYLoad
driver.diagnostic.configure.system.dapi.logging.file.rpc.set_payload(payload =␣
↪False)
```

No command help available

param payload
No help available

set_value(*enable: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC
driver.diagnostic.configure.system.dapi.logging.file.rpc.set_value(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.configure.system.dapi.logging.file.rpc.clone()
```

Subgroups

6.6.2.2.1.6 FilterPy

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:FILTer:MNAMe
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:FILTer:RNAMe
```

class FilterPyCls

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

get_mname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:FILTer:MNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.file.rpc.filterPy.
↪get_mname()
```

No command help available

return
filter_mname: No help available

get_rname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:FILTer:RNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.file.rpc.filterPy.
↪get_rname()
```

No command help available

return
filter_rname: No help available

set_mname(*filter_mname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:FILTer:MNAMe
driver.diagnostic.configure.system.dapi.logging.file.rpc.filterPy.set_
↪mname(filter_mname = 'abc')
```

No command help available

param filter_mname
No help available

set_rname(*filter_rname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:FILE:RPC:FILTer:RNAMe
driver.diagnostic.configure.system.dapi.logging.file.rpc.filterPy.set_
↪rname(filter_rname = 'abc')
```

No command help available

param filter_rname
No help available

6.6.2.2.1.7 Mars

class MarsCls

Mars 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.configure.system.dapi.logging.mars.clone()
```

Subgroups

6.6.2.2.1.8 Psub

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:PAYLoad
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB
```

class PsubCls

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

get_payload() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:PAYLoad
value: bool = driver.diagnostic.configure.system.dapi.logging.mars.psub.get_
↪payload()
```

No command help available

return
payload: No help available

get_value() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB
value: bool = driver.diagnostic.configure.system.dapi.logging.mars.psub.get_
↪value()
```

No command help available

return
enable: No help available

set_payload(*payload: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:PAYLoad
driver.diagnostic.configure.system.dapi.logging.mars.psub.set_payload(payload =␣
↪False)
```

No command help available

param payload
No help available

set_value(*enable: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB
driver.diagnostic.configure.system.dapi.logging.mars.psub.set_value(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.configure.system.dapi.logging.mars.psub.clone()
```

Subgroups

6.6.2.2.1.9 FilterPy

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:FILTer:MNAMe
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:FILTer:RNAMe
```

class FilterPyCls

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

get_mname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:FILTer:MNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.mars.psub.filterPy.
↪get_mname()
```

No command help available

return
filter_mname: No help available

get_rname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:FILTer:RNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.mars.psub.filterPy.
↪get_rname()
```

No command help available

return
filter_rname: No help available

set_mname(*filter_mname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:FILTer:MNAMe
driver.diagnostic.configure.system.dapi.logging.mars.psub.filterPy.set_
↪mname(filter_mname = 'abc')
```

No command help available

param filter_mname
No help available

set_rname(*filter_rname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:PSUB:FILTer:RNAMe
driver.diagnostic.configure.system.dapi.logging.mars.psub.filterPy.set_
↪rname(filter_rname = 'abc')
```

No command help available

param filter_rname
No help available

6.6.2.2.1.10 Rpc

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:PAYLoad
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC
```

class RpcCls

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

get_payload() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:PAYLoad
value: bool = driver.diagnostic.configure.system.dapi.logging.mars.rpc.get_
↪payload()
```

No command help available

return
payload: No help available

get_value() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC
value: bool = driver.diagnostic.configure.system.dapi.logging.mars.rpc.get_
↪value()
```

No command help available

return
enable: No help available

set_payload(*payload: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:PAYLoad
driver.diagnostic.configure.system.dapi.logging.mars.rpc.set_payload(payload =␣
↪False)
```

No command help available

param payload
No help available

set_value(*enable: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC
driver.diagnostic.configure.system.dapi.logging.mars.rpc.set_value(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.configure.system.dapi.logging.mars.rpc.clone()
```

Subgroups

6.6.2.2.1.11 FilterPy

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:FILTer:MNAMe
DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:FILTer:RNAMe
```

class FilterPyCls

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

get_mname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:FILTer:MNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.mars.rpc.filterPy.
↪get_mname()
```

No command help available

return
filter_mname: No help available

get_rname() → str

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:FILTer:RNAMe
value: str = driver.diagnostic.configure.system.dapi.logging.mars.rpc.filterPy.
↪get_rname()
```

No command help available

return
filter_rname: No help available

set_mname(*filter_mname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:FILTer:MNAMe
driver.diagnostic.configure.system.dapi.logging.mars.rpc.filterPy.set_
↪mname(filter_mname = 'abc')
```

No command help available

param filter_mname
No help available

set_rname(*filter_rname: str*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:DAPI:LOGGing:MARS:RPC:FILTer:RNAMe
driver.diagnostic.configure.system.dapi.logging.mars.rpc.filterPy.set_
↪rname(filter_rname = 'abc')
```

No command help available

param filter_rname
No help available

6.6.2.2.2 Scpi

class ScpiCls

Scpi 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.configure.system.scpi.clone()
```

Subgroups

6.6.2.2.2.1 Logging

SCPI Commands :

```
DIAGnostic[:CONFigure]:SYSTem:SCPI:LOGGing:FILE
DIAGnostic[:CONFigure]:SYSTem:SCPI:LOGGing:MARS
```

class LoggingCls

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

get_file() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:SCPI:LOGGing:FILE
value: bool = driver.diagnostic.configure.system.scpi.logging.get_file()
```

No command help available

return
enable: No help available

get_mars() → bool

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:SCPI:LOGGing:MARS
value: bool = driver.diagnostic.configure.system.scpi.logging.get_mars()
```

No command help available

return
enable: No help available

set_file(*enable: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:SCPI:LOGGing:FILE
driver.diagnostic.configure.system.scpi.logging.set_file(enable = False)
```

No command help available

param enable
No help available

set_mars(*enable: bool*) → None

```
# SCPI: DIAGnostic[:CONFigure]:SYSTem:SCPI:LOGGing:MARS
driver.diagnostic.configure.system.scpi.logging.set_mars(enable = False)
```

No command help available

param enable
No help available

6.6.3 Fetch

class FetchCls

Fetch 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.fetch.clone()
```

Subgroups

6.6.3.1 Power

class PowerCls

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

Cloning the Group

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

Subgroups

6.6.3.1.1 State

SCPI Command :

```
DIAGnostic:FETCh:POWer:STATe
```

class StateCls

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

class GetStruct

Response structure. Fields:

- Main_State: enums.TargetStateA: No parameter help available
- Synch_State: enums.TargetSyncState: No parameter help available

get(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → GetStruct

```
# SCPI: DIAGnostic:FETCh:POWer:STATe
value: GetStruct = driver.diagnostic.fetch.power.state.get(timeout = 1.0,␣
↪target_main_state = enums.TargetStateA.OFF, target_sync_state = enums.
↪TargetSyncState.ADJusted)
```

No command help available

param timeout
No help available

param target_main_state
No help available

param target_sync_state
No help available

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

6.6.4 Generic

class GenericCls

Generic 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.generic.clone()
```

Subgroups

6.6.4.1 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.diagnostic.generic.measurement.clone()
```

Subgroups

6.6.4.1.1 Dapi

SCPI Command :

```
DIAGnostic:GENeric:MEASurement:DAPI:TOUT
```

class DapiCls

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

get_timeout() → float

```
# SCPI: DIAGnostic:GENeric:MEASurement:DAPI:TOUT
value: float = driver.diagnostic.generic.measurement.dapi.get_timeout()
```

No command help available

return
timeout: No help available

set_timeout(*timeout: float*) → None

```
# SCPI: DIAGnostic:GENeric:MEASurement:DAPI:TOUT
driver.diagnostic.generic.measurement.dapi.set_timeout(timeout = 1.0)
```

No command help available

param timeout
No help available

6.6.5 Gprf

class GprfCls

Gprf commands group definition. 29 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.5.1 Generator

SCPI Command :

```
DIAGnostic:GPRF:GENerator<Instance>:PNMode
```

class GeneratorCls

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

get_pn_mode() → bool

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:PNMode
value: bool = driver.diagnostic.gprf.generator.get_pn_mode()
```

No command help available

return
pn_mode: No help available

set_pn_mode(*pn_mode: bool*) → None

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:PNMode
driver.diagnostic.gprf.generator.set_pn_mode(pn_mode = False)
```

No command help available

param pn_mode
No help available

Cloning the Group

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

Subgroups

6.6.5.1.1 Correction

SCPI Command :

```
DIAGnostic:GPRF:GENerator<Instance>:CORR
```

class CorrectionCls

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

get() → str

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:CORR
value: str = driver.diagnostic.gprf.generator.correction.get()
```

No command help available

return
corr_cmd_result: No help available

set(*corr_cmd: str*) → None

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:CORR
driver.diagnostic.gprf.generator.correction.set(corr_cmd = 'abc')
```

No command help available

param corr_cmd
No help available

6.6.5.1.2 RfProperty

SCPI Command :

```
DIAGnostic:GPRF:GENerator<Instance>:RFPRoperty:FRANges
```

class RfPropertyCls

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

get_franges() → List[int]

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:RFPRoperty:FRANges
value: List[int] = driver.diagnostic.gprf.generator.rfProperty.get_franges()
```

No command help available

return
value: No help available

6.6.5.1.3 RfSettings

SCPI Command :

```
DIAGnostic:GPRF:GENerator<Instance>:RFSettings:NSMargin
```

class RfSettingsCls

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

get_ns_margin() → float

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:RFSettings:NSMargin
value: float = driver.diagnostic.gprf.generator.rfSettings.get_ns_margin()
```

No command help available

return
net_std_margin: No help available

set_ns_margin(*net_std_margin: float*) → None

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:RFSettings:NSMargin
driver.diagnostic.gprf.generator.rfSettings.set_ns_margin(net_std_margin = 1.0)
```

No command help available

param net_std_margin
No help available

6.6.5.1.4 RfSetttings

SCPI Command :

```
DIAGnostic:GPRF:GENerator<Instance>:RFSetttings:PARatio
```

class RfSetttingsCls

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

get_paratio() → float

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:RFSetttings:PARatio
value: float = driver.diagnostic.gprf.generator.rfSetttings.get_paratio()
```

No command help available

return
peak_avg_ratio: No help available

set_paratio(*peak_avg_ratio: float*) → None

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:RFSetttings:PARatio
driver.diagnostic.gprf.generator.rfSetttings.set_paratio(peak_avg_ratio = 1.0)
```

No command help available

param peak_avg_ratio
No help available

6.6.5.1.5 Rms

SCPI Command :

```
DIAGnostic:GPRF:GENerator<Instance>:RMS:OFFSet
```

class RmsCls

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

get_offset() → float

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:RMS:OFFSet
value: float = driver.diagnostic.gprf.generator.rms.get_offset()
```

No command help available

return
rms_offset: No help available

set_offset(*rms_offset: float*) → None

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:RMS:OFFSet
driver.diagnostic.gprf.generator.rms.set_offset(rms_offset = 1.0)
```

No command help available

param rms_offset
No help available

6.6.5.1.6 Snumber

SCPI Commands :

```
DIAGnostic:GPRF:GENerator<Instance>:SNUMber:BBGenerator
DIAGnostic:GPRF:GENerator<Instance>:SNUMber
```

class SnumberCls

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

get_bb_generator() → int

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:SNUMber:BBGenerator
value: int = driver.diagnostic.gprf.generator.snumber.get_bb_generator()
```

No command help available

return
slot_number: No help available

get_value() → int

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:SNUMber
value: int = driver.diagnostic.gprf.generator.snumber.get_value()
```

No command help available

return
slot_number: No help available

set_bb_generator(*slot_number: int*) → None

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:SNUMber:BBGenerator
driver.diagnostic.gprf.generator.snumber.set_bb_generator(slot_number = 1)
```

No command help available

param slot_number
No help available

set_value(*slot_number: int*) → None

```
# SCPI: DIAGnostic:GPRF:GENerator<Instance>:SNUMber
driver.diagnostic.gprf.generator.snumber.set_value(slot_number = 1)
```

No command help available

param slot_number
No help available

6.6.5.2 Measurement

SCPI Commands :

```
DIAGnostic:GPRF:MEASurement<Instance>:RLEVel
DIAGnostic:GPRF:MEASurement<Instance>:DEBug
DIAGnostic:GPRF:MEASurement:VERSion
```

class MeasurementCls

Measurement commands group definition. 21 total commands, 3 Subgroups, 3 group commands

get_debug() → bool

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:DEBug
value: bool = driver.diagnostic.gprf.measurement.get_debug()
```

No command help available

return
enable: No help available

get_rlevel() → float

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RLEVel
value: float = driver.diagnostic.gprf.measurement.get_rlevel()
```

No command help available

return
reference_level: No help available

get_version() → str

```
# SCPI: DIAGnostic:GPRF:MEASurement:VERSion
value: str = driver.diagnostic.gprf.measurement.get_version()
```

No command help available

return
version: No help available

set_debug(*enable: bool*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:DEBug
driver.diagnostic.gprf.measurement.set_debug(enable = False)
```

No command help available

param enable
No help available

set_rlevel(*reference_level: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RLEVel
driver.diagnostic.gprf.measurement.set_rlevel(reference_level = 1.0)
```

No command help available

param reference_level
No help available

Cloning the Group

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

Subgroups

6.6.5.2.1 Ploss

SCPI Commands :

```
DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:CCALibration
DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:SMODe
```

class PlossCls

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

get_ccalibration() → bool

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:CCALibration
value: bool = driver.diagnostic.gprf.measurement.ploss.get_ccalibration()
```

No command help available

return
calibration: No help available

get_smode() → SelectMode

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:SMODe
value: enums.SelectMode = driver.diagnostic.gprf.measurement.ploss.get_smode()
```

No command help available

return
select_mode: No help available

set_ccalibration(*calibration: bool*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:CCALibration
driver.diagnostic.gprf.measurement.ploss.set_ccalibration(calibration = False)
```

No command help available

param calibration
No help available

Cloning the Group

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

Subgroups

6.6.5.2.1.1 Rnames

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:RNAMes
```

class RnamesCls

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

get() → List[str]

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:RNAMes
value: List[str] = driver.diagnostic.gprf.measurement.ploss.rnames.get()
```

No command help available

return
res_names: No help available

set(*trigger: Trigger*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:PLOSs:RNAMes
driver.diagnostic.gprf.measurement.ploss.rnames.set(trigger = enums.Trigger.
↪CLEarlist)
```

No command help available

param trigger
No help available

6.6.5.2.2 RfProperty

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:TFILter
```

class RfPropertyCls

RfProperty commands group definition. 13 total commands, 10 Subgroups, 1 group commands

get_tfilter() → bool

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:TFILter
value: bool = driver.diagnostic.gprf.measurement.rfProperty.get_tfilter()
```

No command help available

return
def_py: No help available

set_tfilter(*def_py: bool*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:TFILter
driver.diagnostic.gprf.measurement.rfProperty.set_tfilter(def_py = False)
```

No command help available

param def_py
No help available

Cloning the Group

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

Subgroups

6.6.5.2.2.1 Bandpass

class BandpassCls

Bandpass 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.gprf.measurement.rfProperty.bandpass.clone()
```

Subgroups

6.6.5.2.2.2 Bandwidth

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:BANDpass:BWIDth
```

class BandwidthCls

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

class BandwidthStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → BandwidthStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:BANDpass:BWIDth
value: BandwidthStruct = driver.diagnostic.gprf.measurement.rfProperty.bandpass.
↪bandwidth.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:BANDpass:BWIDth
driver.diagnostic.gprf.measurement.rfProperty.bandpass.bandwidth.set(min_py = 1.
↪0, max_py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.3 Franges

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FRANges:MINDex
```

class FrangesCls

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

get_mindex() → int

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FRANges:MINDex
value: int = driver.diagnostic.gprf.measurement.rfProperty.franges.get_mindex()
```

No command help available

return
max_index: No help available

6.6.5.2.2.4 Frequency

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FREQuency
```

class FrequencyCls

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

class FrequencyStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → FrequencyStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FREQuency
value: FrequencyStruct = driver.diagnostic.gprf.measurement.rfProperty.
↪frequency.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FREQuency
driver.diagnostic.gprf.measurement.rfProperty.frequency.set(min_py = 1.0, max_
↪py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.5 Fspan

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FSPan
```

class FspanCls

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

class FspanStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → FspanStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FSPan
value: FspanStruct = driver.diagnostic.gprf.measurement.rfProperty.fspan.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:FSPan
driver.diagnostic.gprf.measurement.rfProperty.fspan.set(min_py = 1.0, max_py =␣
↪1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.6 Gauss

class GaussCls

Gauss 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.gprf.measurement.rfProperty.gauss.clone()
```

Subgroups

6.6.5.2.2.7 Bandwidth

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:GAUSs:BWIDth
```

class BandwidthCls

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

class BandwidthStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → BandwidthStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:GAUSs:BWIDth
value: BandwidthStruct = driver.diagnostic.gprf.measurement.rfProperty.gauss.
↪bandwidth.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:GAUSs:BWIDth
driver.diagnostic.gprf.measurement.rfProperty.gauss.bandwidth.set(min_py = 1.0,␣
↪max_py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.8 IqRecorder

class IqRecorderCls

IqRecorder 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.diagnostic.gprf.measurement.rfProperty.iqRecorder.clone()
```

Subgroups

6.6.5.2.2.9 Bandpass

class BandpassCls

Bandpass 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.gprf.measurement.rfProperty.iqRecorder.bandpass.clone()
```

Subgroups

6.6.5.2.2.10 Bandwidth

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:IQRecorder:BANDpass:BWIDth
```

class BandwidthCls

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

class BandwidthStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → BandwidthStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>
↪:RFPRoperty:IQRecorder:BANDpass:BWIDth
value: BandwidthStruct = driver.diagnostic.gprf.measurement.rfProperty.
↪iqRecorder.bandpass.bandwidth.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>
↪:RFPRoperty:IQRecorder:BANDpass:BWIDth
driver.diagnostic.gprf.measurement.rfProperty.iqRecorder.bandpass.bandwidth.
↪set(min_py = 1.0, max_py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.11 Gauss

class GaussCls

Gauss 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.gprf.measurement.rfProperty.iqRecorder.gauss.clone()
```

Subgroups

6.6.5.2.2.12 Bandwidth

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:IQRecorder:GAUSs:BWIDth
```

class BandwidthCls

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

class BandwidthStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → BandwidthStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:IQRecorder:GAUSs:BWIDth
value: BandwidthStruct = driver.diagnostic.gprf.measurement.rfProperty.
↪iqRecorder.gauss.bandwidth.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:IQRecorder:GAUSs:BWIDth
driver.diagnostic.gprf.measurement.rfProperty.iqRecorder.gauss.bandwidth.
↪set(min_py = 1.0, max_py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.13 Samples

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:IQRecorder:SAMPles
```

class SamplesCls

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

class SamplesStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → SamplesStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:IQRecorder:SAMPles
value: SamplesStruct = driver.diagnostic.gprf.measurement.rfProperty.iqRecorder.
↪samples.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:IQRecorder:SAMPles
driver.diagnostic.gprf.measurement.rfProperty.iqRecorder.samples.set(min_py = 1.
↪0, max_py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.14 ListPy

class ListPyCls

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

Cloning the Group

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

Subgroups

6.6.5.2.2.15 Iranges

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:LIST:IRANges
```

class IrangesCls

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

class IrangesStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available

- Def_Py: float: No parameter help available

get() → IrangesStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:LIST:IRANges
value: IrangesStruct = driver.diagnostic.gprf.measurement.rfProperty.listPy.
↪iranges.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:LIST:IRANges
driver.diagnostic.gprf.measurement.rfProperty.listPy.iranges.set(min_py = 1.0,␣
↪max_py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.16 LoFrequency

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:LOFRequency:AVAilable
```

class LoFrequencyCls

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

get_available() → bool

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:LOFRequency:AVAilable
value: bool = driver.diagnostic.gprf.measurement.rfProperty.loFrequency.get_
↪available()
```

No command help available

return
def_py: No help available

set_available(*def_py: bool*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:LOFRequency:AVAilable
driver.diagnostic.gprf.measurement.rfProperty.loFrequency.set_available(def_py␣
↪= False)
```

No command help available

param def_py
No help available

6.6.5.2.2.17 NbLevel

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:NBLevel
```

class NbLevelCls

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

class NbLevelStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → NbLevelStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:NBLevel
value: NbLevelStruct = driver.diagnostic.gprf.measurement.rfProperty.nbLevel.
↪get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:NBLevel
driver.diagnostic.gprf.measurement.rfProperty.nbLevel.set(min_py = 1.0, max_py␣
↪= 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.2.18 SymbolRate

SCPI Command :

```
DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:SRATe
```

class SymbolRateCls

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

class SymbolRateStruct

Response structure. Fields:

- Min_Py: float: No parameter help available
- Max_Py: float: No parameter help available
- Def_Py: float: No parameter help available

get() → SymbolRateStruct

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:SRATe
value: SymbolRateStruct = driver.diagnostic.gprf.measurement.rfProperty.
↪symbolRate.get()
```

No command help available

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

set(*min_py: float*, *max_py: float*, *def_py: float*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:RFPRoperty:SRATe
driver.diagnostic.gprf.measurement.rfProperty.symbolRate.set(min_py = 1.0, max_
↪py = 1.0, def_py = 1.0)
```

No command help available

param min_py
No help available

param max_py
No help available

param def_py
No help available

6.6.5.2.3 Snumber

SCPI Commands :

```
DIAGnostic:GPRF:MEASurement<Instance>:SNUMber:BBMeas
DIAGnostic:GPRF:MEASurement<Instance>:SNUMber
```

class SnumberCls

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

get_bb_meas() → int

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:SNUMber:BBMeas
value: int = driver.diagnostic.gprf.measurement.snumber.get_bb_meas()
```

No command help available

return
slot_number: No help available

get_value() → int

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:SNUMber
value: int = driver.diagnostic.gprf.measurement.snumber.get_value()
```

No command help available

return
slot_number: No help available

set_bb_meas(*slot_number: int*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:SNUMber:BBMeas
driver.diagnostic.gprf.measurement.snumber.set_bb_meas(slot_number = 1)
```

No command help available

param slot_number
No help available

set_value(*slot_number: int*) → None

```
# SCPI: DIAGnostic:GPRF:MEASurement<Instance>:SNUMber
driver.diagnostic.gprf.measurement.snumber.set_value(slot_number = 1)
```

No command help available

param slot_number
No help available

6.6.6 Meas

class MeasCls

Meas 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.meas.clone()
```

Subgroups

6.6.6.1 Scpi

SCPI Commands :

```
DIAGnostic:MEAS:SCPI:VERSion
DIAGnostic:MEAS:SCPI:HOST
```

class ScpiCls

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

get_host() → str

```
# SCPI: DIAGnostic:MEAS:SCPI:HOST
value: str = driver.diagnostic.meas.scpi.get_host()
```

No command help available

return
name: No help available

get_version() → float

```
# SCPI: DIAGnostic:MEAS:SCPI:VERSion
value: float = driver.diagnostic.meas.scpi.get_version()
```

No command help available

return
version: No help available

6.6.7 Route

class RouteCls

Route commands group definition. 4 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.7.1 Gprf

class GprfCls

Gprf 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.route.gprf.clone()
```

Subgroups

6.6.7.1.1 Generator

SCPI Command :

```
DIAGnostic:ROUTe:GPRF:GENerator<Instance>:SPATh
```

class GeneratorCls

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

get_spath() → List[str]

```
# SCPI: DIAGnostic:ROUTe:GPRF:GENerator<Instance>:SPATh
value: List[str] = driver.diagnostic.route.gprf.generator.get_spath()
```

No command help available

return
signal_path: No help available

set_spath(*signal_path: List[str]*) → None

```
# SCPI: DIAGnostic:ROUTe:GPRF:GENerator<Instance>:SPATh
driver.diagnostic.route.gprf.generator.set_spath(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.6.7.1.2 Measurement

SCPI Command :

```
DIAGnostic:ROUTe:GPRF:MEASurement<Instance>:SPATh
```

class MeasurementCls

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

get_spath() → List[str]

```
# SCPI: DIAGnostic:ROUTe:GPRF:MEASurement<Instance>:SPATh
value: List[str] = driver.diagnostic.route.gprf.measurement.get_spath()
```

No command help available

return
signal_path: No help available

set_spath(*signal_path: List[str]*) → None

```
# SCPI: DIAGnostic:ROUTe:GPRF:MEASurement<Instance>:SPATh
driver.diagnostic.route.gprf.measurement.set_spath(signal_path = ['abc1', 'abc2
↪', 'abc3'])
```

No command help available

param signal_path
No help available

6.6.7.2 NrMmw

class NrMmwCls

NrMmw 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.route.nrMmw.clone()
```

Subgroups

6.6.7.2.1 Measurement

SCPI Command :

```
DIAGnostic:ROUTe:NRMMw:MEASurement<Instance>:SPATh
```

class MeasurementCls

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

get_spath() → List[str]

```
# SCPI: DIAGnostic:ROUTe:NRMMw:MEASurement<Instance>:SPATh
value: List[str] = driver.diagnostic.route.nrMmw.measurement.get_spath()
```

No command help available

return
signal_path: No help available

set_spath(*signal_path: List[str]*) → None

```
# SCPI: DIAGnostic:ROUTe:NRMMw:MEASurement<Instance>:SPATh
driver.diagnostic.route.nrMmw.measurement.set_spath(signal_path = ['abc1', 'abc2
↪', 'abc3'])
```

No command help available

param signal_path
No help available

6.6.7.3 Uwb

class UwbCls

Uwb 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.route.uwb.clone()
```

Subgroups

6.6.7.3.1 Measurement

SCPI Command :

```
DIAGnostic:ROUTe:UWB:MEASurement<Instance>:SPATh
```

class MeasurementCls

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

get_spath() → List[str]

```
# SCPI: DIAGnostic:ROUTe:UWB:MEASurement<Instance>:SPATh
value: List[str] = driver.diagnostic.route.uwb.measurement.get_spath()
```

No command help available

return
signal_path: No help available

set_spath(*signal_path: List[str]*) → None

```
# SCPI: DIAGnostic:ROUTe:UWB:MEASurement<Instance>:SPATh
driver.diagnostic.route.uwb.measurement.set_spath(signal_path = ['abc1', 'abc2',
↪ 'abc3'])
```

No command help available

param signal_path
No help available

6.6.8 Trigger

class TriggerCls

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

Cloning the Group

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

Subgroups

6.6.8.1 Add

class AddCls

Add 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.trigger.add.clone()
```

Subgroups

6.6.8.1.1 Debug

class DebugCls

Debug 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.trigger.add.debug.clone()
```

Subgroups

6.6.8.1.1.1 Output

SCPI Command :

```
DIAGnostic:TRIGger:ADD:DEBug:OUTPut
```

class OutputCls

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

class OutputStruct

Response structure. Fields:

- Show_Trigger_Debug_Output: bool: No parameter help available
- Show_Trigger_Debug_Scpi_Output: bool: No parameter help available

get() → OutputStruct

```
# SCPI: DIAGnostic:TRIGger:ADD:DEBug:OUTPut
value: OutputStruct = driver.diagnostic.trigger.add.debug.output.get()
```

No command help available

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

set(*show_trigger_debug_output: bool*, *show_trigger_debug_scpi_output: bool*) → None

```
# SCPI: DIAGnostic:TRIGger:ADD:DEBug:OUTPut
driver.diagnostic.trigger.add.debug.output.set(show_trigger_debug_output =␣
↪False, show_trigger_debug_scpi_output = False)
```

No command help available

param show_trigger_debug_output
No help available

param show_trigger_debug_scpi_output
No help available

6.7 Gprf

class GprfCls

Gprf commands group definition. 213 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.1 Measurement

class MeasurementCls

Measurement commands group definition. 213 total commands, 9 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.1.1 Canalyzer

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:CANalyzer
STOP:GPRF:MEASurement<Instance>:CANalyzer
ABORt:GPRF:MEASurement<Instance>:CANalyzer
```

class CanalyzerCls

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

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:CANalyzer
driver.gprf.measurement.canalyzer.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:CANalyzer
driver.gprf.measurement.canalyzer.initiate()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
```

(continues on next page)

(continued from previous page)

```
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:CANalyzer
driver.gprf.measurement.canalyzer.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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.gprf.measurement.canalyzer.clone()
```

Subgroups

6.7.1.1.1 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:CANalyzer:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:CANalyzer:STATe
value: enums.ResourceState = driver.gprf.measurement.canalyzer.state.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.canalyzer.state.clone()
```

Subgroups

6.7.1.1.1.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:CANalyzer:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:CANalyzer:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.canalyzer.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.2 ExtPwrSensor

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:EPSensor
STOP:GPRF:MEASurement<Instance>:EPSensor
ABORt:GPRF:MEASurement<Instance>:EPSensor
FETCh:GPRF:MEASurement<Instance>:EPSensor:IDN
FETCh:GPRF:MEASurement<Instance>:EPSensor
READ:GPRF:MEASurement<Instance>:EPSensor
```

class ExtPwrSensorCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator'
- Current_Power: float: No parameter help available
- Average_Power: float: No parameter help available
- Minimum_Power: float: No parameter help available
- Maximum_Power: float: No parameter help available
- Elapsed_Stat: int: Elapsed measurement cycles

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:EPSensor
driver.gprf.measurement.extPwrSensor.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor
value: ResultData = driver.gprf.measurement.extPwrSensor.fetch()
```

Returns all results of the EPS measurement.

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

get_idn() → str

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor:IDN
value: str = driver.gprf.measurement.extPwrSensor.get_idn()
```

Returns the identification string of the connected external sensor.

return
idn: No help available

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:EPSensor
driver.gprf.measurement.extPwrSensor.initiate()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:EPSensor
value: ResultData = driver.gprf.measurement.extPwrSensor.read()
```

Returns all results of the EPS measurement.

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

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:EPSensor
driver.gprf.measurement.extPwrSensor.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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.gprf.measurement.extPwrSensor.clone()
```

Subgroups

6.7.1.2.1 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe
value: enums.ResourceState = driver.gprf.measurement.extPwrSensor.state.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.extPwrSensor.state.clone()
```

Subgroups

6.7.1.2.1.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.extPwrSensor.state.
↪all.fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_
↪sync_state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.3 FftSpecAn

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:FFTSanalyzer
STOP:GPRF:MEASurement<Instance>:FFTSanalyzer
ABORt:GPRF:MEASurement<Instance>:FFTSanalyzer
```

class FftSpecAnCls

FftSpecAn commands group definition. 21 total commands, 5 Subgroups, 3 group commands

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:FFTSanalyzer
driver.gprf.measurement.fftSpecAn.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:FFTSanalyzer
driver.gprf.measurement.fftSpecAn.initiate()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:FFTSanalyzer
driver.gprf.measurement.fftSpecAn.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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.gprf.measurement.fftSpecAn.clone()
```

Subgroups

6.7.1.3.1 Icomponent

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:I
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:I
```

class IcomponentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:I
value: List[float] = driver.gprf.measurement.fftSpecAn.icomponent.fetch()
```

Returns the measured normalized I and Q amplitudes in the time domain.

Suppressed linked return values: reliability

return
idata: Comma-separated list of N normalized I or Q amplitudes. N equals the configured FFT length.

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:I
value: List[float] = driver.gprf.measurement.fftSpecAn.icomponent.read()
```

Returns the measured normalized I and Q amplitudes in the time domain.

Suppressed linked return values: reliability

return
idata: Comma-separated list of N normalized I or Q amplitudes. N equals the configured FFT length.

6.7.1.3.2 Peaks

class PeaksCls

Peaks 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.gprf.measurement.fftSpecAn.peaks.clone()
```

Subgroups

6.7.1.3.2.1 Average

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
```

class AverageCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator'
- Frequency: List[float]: Frequency of the detected peak
- Level: List[float]: Level of the detected peak

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
value: ResultData = driver.gprf.measurement.fftSpecAn.peaks.average.fetch()
```

Returns the results of the peak search in the spectrum diagram. Separate commands retrieve results for the current trace and for the average trace. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}marker 0, ..., {<Frequency>, <Level>}marker 4

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

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
value: ResultData = driver.gprf.measurement.fftSpecAn.peaks.average.read()
```

Returns the results of the peak search in the spectrum diagram. Separate commands retrieve results for the current trace and for the average trace. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}marker 0, ..., {<Frequency>, <Level>}marker 4

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

6.7.1.3.2.2 Current

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
```

class CurrentCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator'
- Frequency: List[float]: Frequency of the detected peak
- Level: List[float]: Level of the detected peak

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
value: ResultData = driver.gprf.measurement.fftSpecAn.peaks.current.fetch()
```

Returns the results of the peak search in the spectrum diagram. Separate commands retrieve results for the current trace and for the average trace. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}marker 0, ..., {<Frequency>, <Level>}marker 4

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

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
value: ResultData = driver.gprf.measurement.fftSpecAn.peaks.current.read()
```

Returns the results of the peak search in the spectrum diagram. Separate commands retrieve results for the current trace and for the average trace. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}marker 0, ..., {<Frequency>, <Level>}marker 4

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

6.7.1.3.3 Power

class PowerCls

Power commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.1.3.3.1 Average

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:AVERage
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:AVERage
value: List[float] = driver.gprf.measurement.fftSpecAn.power.average.fetch()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:AVERage
value: List[float] = driver.gprf.measurement.fftSpecAn.power.average.read()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

6.7.1.3.3.2 Current

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:CURRent
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:CURRent
value: List[float] = driver.gprf.measurement.fftSpecAn.power.current.fetch()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:CURRent
value: List[float] = driver.gprf.measurement.fftSpecAn.power.current.read()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

6.7.1.3.3.3 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MAXimum
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MAXimum
value: List[float] = driver.gprf.measurement.fftSpecAn.power.maximum.fetch()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MAXimum
value: List[float] = driver.gprf.measurement.fftSpecAn.power.maximum.read()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

6.7.1.3.3.4 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MINimum
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MINimum
value: List[float] = driver.gprf.measurement.fftSpecAn.power.minimum.fetch()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MINimum
value: List[float] = driver.gprf.measurement.fftSpecAn.power.minimum.read()
```

Returns the traces of the spectrum diagram. The current, average, minimum and maximum traces can be retrieved. Each trace contains 801 power values and covers the configured frequency span.

Suppressed linked return values: reliability

return
power: Comma-separated list of 801 power values

6.7.1.3.4 Qcomponent

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:Q
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:Q
```

class QcomponentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:Q
value: List[float] = driver.gprf.measurement.fftSpecAn.qcomponent.fetch()
```

Returns the measured normalized I and Q amplitudes in the time domain.

Suppressed linked return values: reliability

return
qdata: Comma-separated list of N normalized I or Q amplitudes. N equals the configured FFT length.

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:Q
value: List[float] = driver.gprf.measurement.fftSpecAn.qcomponent.read()
```

Returns the measured normalized I and Q amplitudes in the time domain.

Suppressed linked return values: reliability

return
qdata: Comma-separated list of N normalized I or Q amplitudes. N equals the configured FFT length.

6.7.1.3.5 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe
value: enums.ResourceState = driver.gprf.measurement.fftSpecAn.state.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.fftSpecAn.state.clone()
```

Subgroups

6.7.1.3.5.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.fftSpecAn.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.4 IqRecorder

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:IQRecorder
ABORt:GPRF:MEASurement<Instance>:IQRecorder
STOP:GPRF:MEASurement<Instance>:IQRecorder
READ:GPRF:MEASurement<Instance>:IQRecorder
FETCh:GPRF:MEASurement<Instance>:IQRecorder
```

class IqRecorderCls

IqRecorder commands group definition. 13 total commands, 5 Subgroups, 5 group commands

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:IQRecorder
driver.gprf.measurement.iqRecorder.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder
value: List[float] = driver.gprf.measurement.iqRecorder.fetch()
```

Returns the I and Q amplitudes in the format specified by FORMat:BASE:DATA. For a detailed description of the data formats, see 'ASCII and binary data formats'. For the number of values n, see method RsCMPX_Gprf.Configure.Gprf. Measurement.IqRecorder.Capture.set.

Suppressed linked return values: reliability

return
iq_samples: For ASCII format: Comma-separated list of I and Q amplitudes {I, Q}1, ..., {I, Q}n For REAL format: Binary block data as listed in the table below. There are no commas within this parameter.

initiate(*save_to_iq_file: FileSave = None*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:IQRecorder
driver.gprf.measurement.iqRecorder.initiate(save_to_iq_file = enums.FileSave.
↪OFF)


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

param save_to_iq_file
Optional parameter, selecting whether the results are written to an I/Q file, to the memory or both. For file selection, see method RsCMPX_Gprf.Configure.Gprf.Measurement.IqRecorder.iqFile. OFF: The results are only stored in the memory. ON: The results are stored in the memory and in a file. ONLY: The results are only stored in a file.

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQRecorder
value: List[float] = driver.gprf.measurement.iqRecorder.read()
```

Returns the I and Q amplitudes in the format specified by FORMat:BASE:DATA. For a detailed description of the data formats, see 'ASCII and binary data formats'. For the number of values n, see method RsCMPX_Gprf.Configure.Gprf. Measurement.IqRecorder.Capture.set.

Suppressed linked return values: reliability

return
iq_samples: For ASCII format: Comma-separated list of I and Q amplitudes {I, Q}1, ..., {I, Q}n For REAL format: Binary block data as listed in the table below. There are no commas within this parameter.

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:IQRecorder
driver.gprf.measurement.iqRecorder.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
```

(continues on next page)

(continued from previous page)

```
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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.gprf.measurement.iqRecorder.clone()
```

Subgroups

6.7.1.4.1 Bin

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:IQRecorder:BIN
FETCh:GPRF:MEASurement<Instance>:IQRecorder:BIN
```

class BinCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:BIN
value: List[float] = driver.gprf.measurement.iqRecorder.bin.fetch()
```

Returns I/Q recorder results in binary format. For the number of values n, see method RsCMPX_Gprf.Configure.Gprf. Measurement.IqRecorder.Capture.set.

return
iq_samples: Binary block data. For a detailed description, see 'ASCII and binary data formats'.

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQRecorder:BIN
value: List[float] = driver.gprf.measurement.iqRecorder.bin.read()
```

Returns I/Q recorder results in binary format. For the number of values n, see method RsCMPX_Gprf.Configure.Gprf. Measurement.IqRecorder.Capture.set.

return
iq_samples: Binary block data. For a detailed description, see 'ASCII and binary data formats'.

6.7.1.4.2 Reliability

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:RELiability
```

class ReliabilityCls

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

fetch() → int

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:RELiability
value: int = driver.gprf.measurement.iqRecorder.reliability.fetch()
```

Queries the reliability indicator for the I/Q recorder, see 'Reliability indicator'.

return
reliability_flag: Two equal values, separated by a comma (e.g. 0,0 for OK)

6.7.1.4.3 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe
value: enums.ResourceState = driver.gprf.measurement.iqRecorder.state.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.iqRecorder.state.clone()
```

Subgroups

6.7.1.4.3.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.iqRecorder.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.4.4 SymbolRate

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:SRATe
```

class SymbolRateCls

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

fetch() → float

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:SRATe
value: float = driver.gprf.measurement.iqRecorder.symbolRate.fetch()
```

Returns the sampling rate of the I/Q recorder, resulting from the filter settings and the configured sample ratio.

return
sample_rate: No help available

6.7.1.4.5 Talignment

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:TALignment
READ:GPRF:MEASurement<Instance>:IQRecorder:TALignment
```

class TalignmentCls

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

fetch() → float

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:TALignment
value: float = driver.gprf.measurement.iqRecorder.talignment.fetch()
```

No command help available

Suppressed linked return values: reliability

return
time_alignment: No help available

read() → float

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQRecorder:TALignment
value: float = driver.gprf.measurement.iqRecorder.talignment.read()
```

No command help available

Suppressed linked return values: reliability

return
time_alignment: No help available

6.7.1.5 IqVsSlot

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:IQVSlot
STOP:GPRF:MEASurement<Instance>:IQVSlot
ABORt:GPRF:MEASurement<Instance>:IQVSlot
```

class IqVsSlotCls

IqVsSlot commands group definition. 18 total commands, 7 Subgroups, 3 group commands

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:IQVSlot
driver.gprf.measurement.iqVsSlot.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:IQVSlot
driver.gprf.measurement.iqVsSlot.initiate()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:IQVSlot
driver.gprf.measurement.iqVsSlot.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
```

(continues on next page)

(continued from previous page)

```
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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.gprf.measurement.iqVsSlot.clone()
```

Subgroups

6.7.1.5.1 FreqError

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:IQVSlot:FERRor
FETCh:GPRF:MEASurement<Instance>:IQVSlot:FERRor
```

class FreqErrorCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:FERRor
value: List[float] = driver.gprf.measurement.iqVsSlot.freqError.fetch()
```

Returns the contents of the frequency error result diagram.

Suppressed linked return values: reliability

return
frequency_error: Comma-separated list of frequency errors, one value per measured step

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:FERRor
value: List[float] = driver.gprf.measurement.iqVsSlot.freqError.read()
```

Returns the contents of the frequency error result diagram.

Suppressed linked return values: reliability

return
frequency_error: Comma-separated list of frequency errors, one value per measured step

6.7.1.5.2 Icomponent

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:IQVSlot:I
FETCh:GPRF:MEASurement<Instance>:IQVSlot:I
```

class IcomponentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:I
value: List[float] = driver.gprf.measurement.iqVsSlot.icomponent.fetch()
```

Returns the contents of the I and Q result diagrams.

Suppressed linked return values: reliability

return
idata: Comma-separated list of amplitudes, one value per measured step

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:I
value: List[float] = driver.gprf.measurement.iqVsSlot.icomponent.read()
```

Returns the contents of the I and Q result diagrams.

Suppressed linked return values: reliability

return
idata: Comma-separated list of amplitudes, one value per measured step

6.7.1.5.3 Level

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:IQVSlot:LEVel
FETCh:GPRF:MEASurement<Instance>:IQVSlot:LEVel
```

class LevelCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:LEVel
value: List[float] = driver.gprf.measurement.iqVsSlot.level.fetch()
```

Returns the contents of the level result diagram.

Suppressed linked return values: reliability

return
level: Comma-separated list of levels, one value per measured step

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:LEVel
value: List[float] = driver.gprf.measurement.iqVsSlot.level.read()
```

Returns the contents of the level result diagram.

Suppressed linked return values: reliability

return
level: Comma-separated list of levels, one value per measured step

6.7.1.5.4 OfError

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:IQVSlot:OFERror
READ:GPRF:MEASurement<Instance>:IQVSlot:OFERror
FETCh:GPRF:MEASurement<Instance>:IQVSlot:OFERror
```

class OfErrorCls

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

calculate() → ResultStatus2

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:IQVSlot:OFERror
value: enums.ResultStatus2 = driver.gprf.measurement.iqVsSlot.ofError.
↪calculate()
```

Returns the overall frequency error. The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
frequency_error: Overall frequency error, the arithmetic mean value of the frequency errors of all considered steps.

fetch() → float

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:OFERror
value: float = driver.gprf.measurement.iqVsSlot.ofError.fetch()
```

Returns the overall frequency error. The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
frequency_error: Overall frequency error, the arithmetic mean value of the frequency errors of all considered steps.

read() → float

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:OFERror
value: float = driver.gprf.measurement.iqVsSlot.ofError.read()
```

Returns the overall frequency error. The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
frequency_error: Overall frequency error, the arithmetic mean value of the frequency errors of all considered steps.

6.7.1.5.5 Phase

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:IQVSlot:PHASe
FETCh:GPRF:MEASurement<Instance>:IQVSlot:PHASe
```

class PhaseCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:PHASe
value: List[float] = driver.gprf.measurement.iqVsSlot.phase.fetch()
```

Returns the contents of the phase result diagram.

Suppressed linked return values: reliability

return
phase: Comma-separated list of phases, one value per measured step

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:PHASe
value: List[float] = driver.gprf.measurement.iqVsSlot.phase.read()
```

Returns the contents of the phase result diagram.

Suppressed linked return values: reliability

return
phase: Comma-separated list of phases, one value per measured step

6.7.1.5.6 Qcomponent

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:IQVSlot:Q
FETCh:GPRF:MEASurement<Instance>:IQVSlot:Q
```

class QcomponentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:Q
value: List[float] = driver.gprf.measurement.iqVsSlot.qcomponent.fetch()
```

Returns the contents of the I and Q result diagrams.

Suppressed linked return values: reliability

return
qdata: Comma-separated list of amplitudes, one value per measured step

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:Q
value: List[float] = driver.gprf.measurement.iqVsSlot.qcomponent.read()
```

Returns the contents of the I and Q result diagrams.

Suppressed linked return values: reliability

return
qdata: Comma-separated list of amplitudes, one value per measured step

6.7.1.5.7 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:IQVSlot:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:STATe
value: enums.ResourceState = driver.gprf.measurement.iqVsSlot.state.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.iqVsSlot.state.clone()
```

Subgroups

6.7.1.5.7.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:IQVSlot:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.iqVsSlot.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.6 Nrpm

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:NRPM
STOP:GPRF:MEASurement<Instance>:NRPM
ABORt:GPRF:MEASurement<Instance>:NRPM
```

class NrpmCls

Nrpm commands group definition. 8 total commands, 2 Subgroups, 3 group commands

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:NRPM
driver.gprf.measurement.nrpm.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:NRPM
driver.gprf.measurement.nrpm.initiate()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:NRPM
driver.gprf.measurement.nrpm.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
```

(continues on next page)

(continued from previous page)

```
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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.gprf.measurement.nrpm.clone()
```

Subgroups

6.7.1.6.1 Sensor<Sensor>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.gprf.measurement.nrpm.sensor.repcap_sensor_get()
driver.gprf.measurement.nrpm.sensor.repcap_sensor_set(repcap.Sensor.Nr1)
```

class SensorCls

Sensor commands group definition. 3 total commands, 1 Subgroups, 0 group commands Repeated Capability: Sensor, default value after init: Sensor.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.nrpm.sensor.clone()
```

Subgroups

6.7.1.6.1.1 Power

SCPI Commands :

```
READ:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:POWer
FETCh:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:POWer
CALCulate:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:POWer
```

class PowerCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator'
- State_Antenna_1: enums.ResultStatus2: No parameter help available
- State_Antenna_2: enums.ResultStatus2: No parameter help available
- State_Antenna_3: enums.ResultStatus2: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator'
- Power_Antenna_1: float: Power measured at antenna 1 of the sensor
- Power_Antenna_2: float: Power measured at antenna 2 of the sensor
- Power_Antenna_3: float: Power measured at antenna 3 of the sensor

calculate(*sensor=Sensor.Default*) → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:POWer
value: CalculateStruct = driver.gprf.measurement.nrpm.sensor.power.
↪calculate(sensor = repcap.Sensor.Default)
```

Returns the measurement results for the power sensor connected to Sensor <no>. The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

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

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

fetch(*sensor=Sensor.Default*) → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:POWer
value: ResultData = driver.gprf.measurement.nrpm.sensor.power.fetch(sensor =␣
↪repcap.Sensor.Default)
```

Returns the measurement results for the power sensor connected to Sensor <no>. The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

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

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

read(*sensor=Sensor.Default*) → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:POWer
value: ResultData = driver.gprf.measurement.nrpm.sensor.power.read(sensor =␣
↪repcap.Sensor.Default)
```

Returns the measurement results for the power sensor connected to Sensor <no>. The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

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

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

6.7.1.6.2 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:NRPM:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRPM:STATe
value: enums.ResourceState = driver.gprf.measurement.nrpm.state.fetch(timeout =␣
↪1.0, target_main_state = enums.TargetStateA.OFF, target_sync_state = enums.
↪TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.nrpm.state.clone()
```

Subgroups

6.7.1.6.2.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:NRPM:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRPM:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.nrpm.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.7 Ploss

SCPI Commands :

```
STOP:GPRF:MEASurement<Instance>:PLOSs
ABORt:GPRF:MEASurement<Instance>:PLOSs
```

class PlossCls

Ploss commands group definition. 21 total commands, 7 Subgroups, 2 group commands

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:PLOSs
driver.gprf.measurement.ploss.abort()


    INTRO_CMD_HELP: Stops or aborts the measurement:

    - STOP...: The measurement enters the 'RDY' state. The resources remain␣
```

(continues on next page)

(continued from previous page)

```
↪allocated to the measurement.
    - ABORt...: The measurement enters the 'OFF' state. Allocated resources are␣
↪released.

    :param opc_timeout_ms: Maximum time to wait in milliseconds, valid only for␣
↪this call.
```

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:PLOSs
driver.gprf.measurement.ploss.stop()


    INTRO_CMD_HELP: Stops or aborts the measurement:

    - STOP...: The measurement enters the 'RDY' state. The resources remain␣
↪allocated to the measurement.
    - ABORt...: The measurement enters the 'OFF' state. Allocated resources are␣
↪released.

    :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.gprf.measurement.ploss.clone()
```

Subgroups

6.7.1.7.1 Clear

SCPI Command :

```
INITiate:GPRF:MEASurement<Instance>:PLOSs:CLEar
```

class ClearCls

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

initiate() → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:CLEar
driver.gprf.measurement.ploss.clear.initiate()
```

Discards all measurement results.

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:CLEar
driver.gprf.measurement.ploss.clear.initiate_with_opc()
```

Discards all measurement results.

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

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

6.7.1.7.2 Eeprom

class EepromCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.ploss.eeprom.clone()
```

Subgroups

6.7.1.7.2.1 Eoo

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:EOO
```

class EooCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:EOO
value: List[float] = driver.gprf.measurement.ploss.eeprom.eoo.fetch()
```

No command help available

Suppressed linked return values: reliability

return
iq_samples: No help available

6.7.1.7.2.2 Ezo

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:EZO
```

class EzoCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:EZO
value: List[float] = driver.gprf.measurement.ploss.eeprom.ezo.fetch()
```

No command help available

Suppressed linked return values: reliability

return
iq_samples: No help available

6.7.1.7.2.3 Ezz

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:EZZ
```

class EzzCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:EZZ
value: List[float] = driver.gprf.measurement.ploss.eeprom.ezz.fetch()
```

No command help available

Suppressed linked return values: reliability

return
iq_samples: No help available

6.7.1.7.2.4 Frequency

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:FREQuency
```

class FrequencyCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EEPRom:FREQuency
value: List[float] = driver.gprf.measurement.ploss.eeprom.frequency.fetch()
```

No command help available

Suppressed linked return values: reliability

return
frequency: No help available

6.7.1.7.3 Evaluate

SCPI Command :

```
INITiate:GPRF:MEASurement<instance>:PLOSs:EVALuate
```

class EvaluateCls

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

initiate() → None

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:EVALuate
driver.gprf.measurement.ploss.evaluate.initiate()
```

Selects a measurement step according to the last mnemonic (Open, Short or Evaluation) and starts the measurement.

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

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:EVALuate
driver.gprf.measurement.ploss.evaluate.initiate_with_opc()
```

Selects a measurement step according to the last mnemonic (Open, Short or Evaluation) and starts the measurement.

Same as initiate, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.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.gprf.measurement.ploss.evaluate.clone()
```

Subgroups

6.7.1.7.3.1 Frequency

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:FREQuency
```

class FrequencyCls

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

fetch(*connection_name: str*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:FREQuency
value: List[float] = driver.gprf.measurement.ploss.evaluate.frequency.
↪fetch(connection_name = 'abc')
```

Returns the frequency values of the result table for a selected RF connection. The order of the list entries is the same as in the command method RsCMPX_Gprf.Gprf.Measurement.Ploss.Evaluate.Gain.fetch. Use this command to check at which frequencies the gain values have been measured. For possible connection names, see method RsCMPX_Gprf.Catalog.Gprf. Measurement.Ploss.cname.

Suppressed linked return values: reliability

param connection_name
RF connection for which results are queried.

return
frequency: Comma-separated list of frequency values.

6.7.1.7.3.2 Gain

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:GAIN
```

class GainCls

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

fetch(*connection_name: str*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:GAIN
value: List[float] = driver.gprf.measurement.ploss.evaluate.gain.
↪fetch(connection_name = 'abc')
```

Returns the gain values of the result table for a selected RF connection. For possible connection names, see method RsCMPX_Gprf.Catalog.Gprf.Measurement.Ploss.cname.

Suppressed linked return values: reliability

param connection_name
RF connection for which results are queried.

return
gain: Comma-separated list of gain values.

6.7.1.7.3.3 State

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:STATe
```

class StateCls

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

fetch(*connection_name: str*) → PathLossState

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:STATe
value: enums.PathLossState = driver.gprf.measurement.ploss.evaluate.state.
↪fetch(connection_name = 'abc')
```

Queries the result state for the measurement step Evaluation and a selected RF connection. For possible connection names, see method RsCMPX_Gprf.Catalog.Gprf.Measurement.Ploss.cname.

Suppressed linked return values: reliability

param connection_name
RF connection for which the result state is queried.

return
result_state: NCAP: No measurement results available. PEND: Measurement is running. RDY: Measurement is complete, results available.

6.7.1.7.3.4 Trace

class TraceCls

Trace 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.gprf.measurement.ploss.evaluate.trace.clone()
```

Subgroups

6.7.1.7.3.5 Frequency

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:TRACe:FREQuency
```

class FrequencyCls

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

fetch(*connection_name: str*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:TRACe:FREQuency
value: List[float] = driver.gprf.measurement.ploss.evaluate.trace.frequency.
↪fetch(connection_name = 'abc')
```

Returns the frequency values of the result diagram for a selected RF connection. The order of the values is the same as in the command method RsCMPX_Gprf.Gprf.Measurement.Ploss.Evaluate.Trace.Gain.fetch. Use this command to check at which frequencies the gain values have been measured. For possible connection names, see method RsCMPX_Gprf.Catalog.Gprf. Measurement.Ploss.cname.

Suppressed linked return values: reliability

param connection_name
RF connection for which results are queried.

return
frequency: Comma-separated list of frequency values.

6.7.1.7.3.6 Gain

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:TRACe:GAIN
```

class GainCls

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

fetch(*connection_name: str*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:EVALuate:TRACe:GAIN
value: List[float] = driver.gprf.measurement.ploss.evaluate.trace.gain.
↪fetch(connection_name = 'abc')
```

Returns the gain values of the result diagram for a selected RF connection. For possible connection names, see method RsCMPX_Gprf.Catalog.Gprf.Measurement.Ploss.cname.

Suppressed linked return values: reliability

param connection_name
RF connection for which results are queried.

return
gain: Comma-separated list of gain values.

6.7.1.7.4 Match

SCPI Command :

```
INITiate:GPRF:MEASurement<instance>:PLOSs:MATCh
```

class MatchCls

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

initiate() → None

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:MATCh
driver.gprf.measurement.ploss.match.initiate()
```

No command help available

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

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:MATCh
driver.gprf.measurement.ploss.match.initiate_with_opc()
```

No command help available

Same as initiate, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.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.gprf.measurement.ploss.match.clone()
```

Subgroups

6.7.1.7.4.1 State

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:MATCh:STATe
```

class StateCls

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

fetch(*connection_name: str*) → PathLossState

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:MATCh:STATe
value: enums.PathLossState = driver.gprf.measurement.ploss.match.state.
↪fetch(connection_name = 'abc')
```

No command help available

Suppressed linked return values: reliability

param connection_name
No help available

return
result_state: No help available

6.7.1.7.5 Open

SCPI Command :

```
INITiate:GPRF:MEASurement<instance>:PLOSs:OPEN
```

class OpenCls

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

initiate() → None

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:OPEN
driver.gprf.measurement.ploss.open.initiate()
```

Selects a measurement step according to the last mnemonic (Open, Short or Evaluation) and starts the measurement.

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

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:OPEN
driver.gprf.measurement.ploss.open.initiate_with_opc()
```

Selects a measurement step according to the last mnemonic (Open, Short or Evaluation) and starts the measurement.

Same as initiate, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.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.gprf.measurement.ploss.open.clone()
```

Subgroups

6.7.1.7.5.1 State

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:OPEN:STATe
```

class StateCls

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

fetch(*connection_name: str*) → PathLossState

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:OPEN:STATe
value: enums.PathLossState = driver.gprf.measurement.ploss.open.state.
↪fetch(connection_name = 'abc')
```

Queries the result state for the measurement step Open and a selected RF connection. For possible connection names, see method RsCMPX_Gprf.Catalog.Gprf.Measurement.Ploss.cname.

Suppressed linked return values: reliability

param connection_name
RF connection for which the result state is queried.

return
result_state: NCAP: No measurement results available. PEND: Measurement is running. RDY: Measurement is complete, results available.

6.7.1.7.6 Short

SCPI Command :

```
INITiate:GPRF:MEASurement<instance>:PLOSs:SHORt
```

class ShortCls

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

initiate() → None

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:SHORt
driver.gprf.measurement.ploss.short.initiate()
```

Selects a measurement step according to the last mnemonic (Open, Short or Evaluation) and starts the measurement.

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

```
# SCPI: INITiate:GPRF:MEASurement<instance>:PLOSs:SHORt
driver.gprf.measurement.ploss.short.initiate_with_opc()
```

Selects a measurement step according to the last mnemonic (Open, Short or Evaluation) and starts the measurement.

Same as initiate, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.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.gprf.measurement.ploss.short.clone()
```

Subgroups

6.7.1.7.6.1 State

SCPI Command :

```
FETCh:GPRF:MEASurement<instance>:PLOSs:SHORt:STATe
```

class StateCls

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

fetch(*connection_name: str*) → PathLossState

```
# SCPI: FETCh:GPRF:MEASurement<instance>:PLOSs:SHORt:STATe
value: enums.PathLossState = driver.gprf.measurement.ploss.short.state.
↪fetch(connection_name = 'abc')
```

Queries the result state for the measurement step Short and a selected RF connection. For possible connection names, see method RsCMPX_Gprf.Catalog.Gprf.Measurement.Ploss.cname.

Suppressed linked return values: reliability

param connection_name
RF connection for which the result state is queried.

return
result_state: NCAP: No measurement results available. PEND: Measurement is running. RDY: Measurement is complete, results available.

6.7.1.7.7 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:STATe
value: enums.ResourceState = driver.gprf.measurement.ploss.state.fetch(timeout␣
↪= 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_state = enums.
↪TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.ploss.state.clone()
```

Subgroups

6.7.1.7.7.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.ploss.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.8 Power

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:POWer
STOP:GPRF:MEASurement<Instance>:POWer
ABORt:GPRF:MEASurement<Instance>:POWer
```

class PowerCls

Power commands group definition. 71 total commands, 13 Subgroups, 3 group commands

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:POWer
driver.gprf.measurement.power.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:POWer
driver.gprf.measurement.power.initiate()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:POWer
driver.gprf.measurement.power.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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.gprf.measurement.power.clone()
```

Subgroups

6.7.1.8.1 AmplitudeProbDensity

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:APD
```

class AmplitudeProbDensityCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:APD
value: List[float] = driver.gprf.measurement.power.amplitudeProbDensity.fetch()
```

Returns the APD diagram contents.

Suppressed linked return values: reliability

return
results: 4096 results, each representing a 0.047-dB interval (‘bin’)

6.7.1.8.2 Average

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:AVERage
FETCh:GPRF:MEASurement<Instance>:POWer:AVERage
READ:GPRF:MEASurement<Instance>:POWer:AVERage
```

class AverageCls

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

calculate() → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.average.
↪calculate()
```

Returns power results for all segments, see ‘Results in list mode’.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_average_rms: Comma-separated list of power values, one value per measured segment

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[float] = driver.gprf.measurement.power.average.fetch()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_average_rms: Comma-separated list of power values, one value per measured segment

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[float] = driver.gprf.measurement.power.average.read()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_average_rms: Comma-separated list of power values, one value per measured segment

Cloning the Group

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

Subgroups

6.7.1.8.2.1 Rms

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:AVERage:RMS
READ:GPRF:MEASurement<Instance>:POWer:AVERage:RMS
```

class RmsCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:AVERage:RMS
value: List[float] = driver.gprf.measurement.power.average.rms.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power_average_rms: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:AVERage:RMS
value: List[float] = driver.gprf.measurement.power.average.rms.read()
```

No command help available

Suppressed linked return values: reliability

return
power_average_rms: No help available

6.7.1.8.3 CumulativeDistribFnc

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:CCDF
```

class CumulativeDistribFncCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CCDF
value: List[float] = driver.gprf.measurement.power.cumulativeDistribFnc.fetch()
```

Returns the CCDF diagram contents.

Suppressed linked return values: reliability

return
results: 4096 results, each representing a 0.047-dB interval (‘bin’)

Cloning the Group

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

Subgroups

6.7.1.8.3.1 Power

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:POWer
```

class PowerCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: See ‘Reliability indicator’
- Avg: float: Average power of all samples
- Max_Py: float: Maximum power of all samples
- Par: float: Peak to average ratio
- Index_Avg_Power: int: Index of the average power ‘bin’

fetch() → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:POWer
value: FetchStruct = driver.gprf.measurement.power.cumulativeDistribFnc.power.
↪fetch()
```

Returns some statistic results for the power samples.

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

6.7.1.8.3.2 Probability

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:PROBability
```

class ProbabilityCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:PROBability
value: List[float] = driver.gprf.measurement.power.cumulativeDistribFnc.
↪probability.fetch()
```

Returns power values with a certain probability, taken from the CCDF diagram.

Suppressed linked return values: reliability

return
probability: Comma-separated list of 6 power values with the following probabilities: 10 %, 1 %, 0.1 %, 0.01 %, 0.001 %, 0.0001 % The power values are indicated in dB relative to the average power.

6.7.1.8.3.3 Sample

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:SAMPle
```

class SampleCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator'
- Count: int: Total sample count
- Time: float: Total sample time

fetch() → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:SAMPle
value: FetchStruct = driver.gprf.measurement.power.cumulativeDistribFnc.sample.
↪fetch()
```

Returns the sample counters for the APD and CCDF results.

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

6.7.1.8.4 Current

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:CURRent
FETCh:GPRF:MEASurement<Instance>:POWer:CURRent
READ:GPRF:MEASurement<Instance>:POWer:CURRent
```

class CurrentCls

Current commands group definition. 9 total commands, 3 Subgroups, 3 group commands

calculate() → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.current.
↪calculate()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_rms: Comma-separated list of power values, one value per measured segment

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[float] = driver.gprf.measurement.power.current.fetch()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)

- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_rms: Comma-separated list of power values, one value per measured segment

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[float] = driver.gprf.measurement.power.current.read()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_rms: Comma-separated list of power values, one value per measured segment

Cloning the Group

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

Subgroups

6.7.1.8.4.1 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:CURRent:MAXimum
READ:GPRF:MEASurement<Instance>:POWer:CURRent:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CURRent:MAXimum
value: List[float] = driver.gprf.measurement.power.current.maximum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power_current_max: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent:MAXimum
value: List[float] = driver.gprf.measurement.power.current.maximum.read()
```

No command help available

Suppressed linked return values: reliability

return
power_current_max: No help available

6.7.1.8.4.2 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:CURRent:MINimum
READ:GPRF:MEASurement<Instance>:POWer:CURRent:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CURRent:MINimum
value: List[float] = driver.gprf.measurement.power.current.minimum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power_current_min: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent:MINimum
value: List[float] = driver.gprf.measurement.power.current.minimum.read()
```

No command help available

Suppressed linked return values: reliability

return
power_current_min: No help available

6.7.1.8.4.3 Rms

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:CURRent:RMS
READ:GPRF:MEASurement<Instance>:POWer:CURRent:RMS
```

class RmsCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CURRent:RMS
value: List[float] = driver.gprf.measurement.power.current.rms.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power_current_rms: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent:RMS
value: List[float] = driver.gprf.measurement.power.current.rms.read()
```

No command help available

Suppressed linked return values: reliability

return
power_current_rms: No help available

6.7.1.8.5 ElapsedStats

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:ESTatistics
```

class ElapsedStatsCls

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

fetch() → int

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:ESTatistics
value: int = driver.gprf.measurement.power.elapsedStats.fetch()
```

Returns the reliability indicator and the number of elapsed measurement intervals.

Suppressed linked return values: reliability

return
stat_count: Number of elapsed measurement intervals.

6.7.1.8.6 IqData

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:IQData
READ:GPRF:MEASurement<Instance>:POWer:IQData
```

class IqDataCls

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

fetch(*list_index: int*, *result_index: int = None*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:IQData
value: List[float] = driver.gprf.measurement.power.iqData.fetch(list_index = 1,␣
↪result_index = 1)
```

No command help available

Suppressed linked return values: reliability

param list_index
No help available

param result_index
No help available

return
iq_data: No help available

read(*list_index: int*, *result_index: int = None*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:IQData
value: List[float] = driver.gprf.measurement.power.iqData.read(list_index = 1,␣
↪result_index = 1)
```

No command help available

Suppressed linked return values: reliability

param list_index
No help available

param result_index
No help available

return
iq_data: No help available

Cloning the Group

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

Subgroups

6.7.1.8.6.1 Bin

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:IQData:BIN
```

class BinCls

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

fetch(*list_index: int*, *result_index: int = None*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:IQData:BIN
value: List[float] = driver.gprf.measurement.power.iqData.bin.fetch(list_index␣
↪= 1, result_index = 1)
```

No command help available

Suppressed linked return values: reliability

param list_index
No help available

param result_index
No help available

return
iq_data: No help available

6.7.1.8.7 IqInfo

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:IQINfo
```

class IqInfoCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Number_Of_Samples: float: No parameter help available
- Sample_Rate: float: No parameter help available

fetch(*list_index: int*, *result_index: int = None*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:IQINfo
value: FetchStruct = driver.gprf.measurement.power.iqInfo.fetch(list_index = 1,␣
↪result_index = 1)
```

No command help available

param list_index
No help available

param result_index
No help available

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

6.7.1.8.8 ListPy

class ListPyCls

ListPy commands group definition. 21 total commands, 6 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.7.1.8.8.1 Average

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
FETCh:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
READ:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
```

class AverageCls

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

calculate(*list_index: int*) → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.listPy.average.
↪calculate(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.

INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_average_rms: Power value for the selected segment

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
value: List[float] = driver.gprf.measurement.power.listPy.average.fetch(list_
↪index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.

INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)

- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_average_rms: Power value for the selected segment

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
value: List[float] = driver.gprf.measurement.power.listPy.average.read(list_
↪index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_average_rms: Power value for the selected segment

6.7.1.8.8.2 Current

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
FETCh:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
READ:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
```

class CurrentCls

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

calculate(*list_index: int*) → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.listPy.current.
↪calculate(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.

INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_rms: Power value for the selected segment

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
value: List[float] = driver.gprf.measurement.power.listPy.current.fetch(list_
↪index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.

INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)

- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_rms: Power value for the selected segment

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
value: List[float] = driver.gprf.measurement.power.listPy.current.read(list_
↪index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_rms: Power value for the selected segment

6.7.1.8.8.3 Maximum

class MaximumCls

Maximum 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.gprf.measurement.power.listPy.maximum.clone()
```

Subgroups

6.7.1.8.8.4 Current

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
FETCh:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
READ:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
```

class CurrentCls

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

calculate(*list_index: int*) → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.listPy.maximum.
↪current.calculate(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.

INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_max: Power value for the selected segment

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
value: List[float] = driver.gprf.measurement.power.listPy.maximum.current.
↪fetch(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_max: Power value for the selected segment

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
value: List[float] = driver.gprf.measurement.power.listPy.maximum.current.
↪read(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_max: Power value for the selected segment

6.7.1.8.8.5 Minimum

class MinimumCls

Minimum 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.gprf.measurement.power.listPy.minimum.clone()
```

Subgroups

6.7.1.8.8.6 Current

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
FETCh:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
READ:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
```

class CurrentCls

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

calculate(*list_index: int*) → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.listPy.minimum.
↪current.calculate(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)

- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_min: Power value for the selected segment

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
value: List[float] = driver.gprf.measurement.power.listPy.minimum.current.
↪fetch(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_min: Power value for the selected segment

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
value: List[float] = driver.gprf.measurement.power.listPy.minimum.current.
↪read(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)

- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_current_min: Power value for the selected segment

6.7.1.8.8.7 Peak

class PeakCls

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

Cloning the Group

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

Subgroups

6.7.1.8.8.8 Maximum

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
FETCh:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
READ:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
```

class MaximumCls

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

calculate(*list_index: int*) → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.listPy.peak.
↪maximum.calculate(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)

- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_maximum_max: Power value for the selected segment

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
value: List[float] = driver.gprf.measurement.power.listPy.peak.maximum.
↪fetch(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_maximum_max: Power value for the selected segment

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
value: List[float] = driver.gprf.measurement.power.listPy.peak.maximum.
↪read(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_maximum_max: Power value for the selected segment

6.7.1.8.8.9 Minimum

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
FETCh:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
READ:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
```

class MinimumCls

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

calculate(*list_index: int*) → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.listPy.peak.
↪minimum.calculate(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_minimum_min: Power value for the selected segment

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
value: List[float] = driver.gprf.measurement.power.listPy.peak.minimum.
↪fetch(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_minimum_min: Power value for the selected segment

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
value: List[float] = driver.gprf.measurement.power.listPy.peak.minimum.
↪read(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)

- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_minimum_min: Power value for the selected segment

6.7.1.8.8.10 StandardDev

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
FETCh:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
READ:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
```

class StandardDevCls

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

calculate(*list_index: int*) → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.listPy.
↪standardDev.calculate(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_std_dev_cur: Power value for the selected segment

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
value: List[float] = driver.gprf.measurement.power.listPy.standardDev.
↪fetch(list_index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_std_dev_cur: Power value for the selected segment

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
value: List[float] = driver.gprf.measurement.power.listPy.standardDev.read(list_
↪index = 1)
```

Returns power results for segment <ListIndex>, see 'Results in list mode'.
INTRO_CMD_HELP: The following powers can be retrieved:

- Current RMS (...:LIST:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

param list_index
Index of the segment

return
power_std_dev_cur: Power value for the selected segment

6.7.1.8.9 Maximum

class MaximumCls

Maximum 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.gprf.measurement.power.maximum.clone()
```

Subgroups

6.7.1.8.9.1 Current

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
FETCh:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
READ:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
```

class CurrentCls

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

calculate() → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.maximum.
↪current.calculate()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_max: Comma-separated list of power values, one value per measured segment

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[float] = driver.gprf.measurement.power.maximum.current.fetch()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_max: Comma-separated list of power values, one value per measured segment

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[float] = driver.gprf.measurement.power.maximum.current.read()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_max: Comma-separated list of power values, one value per measured segment

6.7.1.8.9.2 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:MAXimum:MAXimum
READ:GPRF:MEASurement<Instance>:POWer:MAXimum:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MAXimum:MAXimum
value: List[float] = driver.gprf.measurement.power.maximum.maximum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power_maximum_max: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MAXimum:MAXimum
value: List[float] = driver.gprf.measurement.power.maximum.maximum.read()
```

No command help available

Suppressed linked return values: reliability

return
power_maximum_max: No help available

6.7.1.8.10 Minimum

class MinimumCls

Minimum 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.gprf.measurement.power.minimum.clone()
```

Subgroups

6.7.1.8.10.1 Current

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
FETCh:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
READ:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
```

class CurrentCls

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

calculate() → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.minimum.
↪current.calculate()
```

Returns power results for all segments, see 'Results in list mode'.

INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return

power_current_min: Comma-separated list of power values, one value per measured segment

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[float] = driver.gprf.measurement.power.minimum.current.fetch()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_min: Comma-separated list of power values, one value per measured segment

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[float] = driver.gprf.measurement.power.minimum.current.read()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_current_min: Comma-separated list of power values, one value per measured segment

6.7.1.8.10.2 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:MINimum:MINimum
READ:GPRF:MEASurement<Instance>:POWer:MINimum:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MINimum:MINimum
value: List[float] = driver.gprf.measurement.power.minimum.minimum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power_minimum_min: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MINimum:MINimum
value: List[float] = driver.gprf.measurement.power.minimum.minimum.read()
```

No command help available

Suppressed linked return values: reliability

return
power_minimum_min: No help available

6.7.1.8.11 Peak

class PeakCls

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

Cloning the Group

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

Subgroups

6.7.1.8.11.1 Maximum

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
READ:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
```

class MaximumCls

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

calculate() → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.peak.maximum.
↪calculate()
```

Returns power results for all segments, see 'Results in list mode'.

INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return

power_maximum_max: Comma-separated list of power values, one value per measured segment

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[float] = driver.gprf.measurement.power.peak.maximum.fetch()
```

Returns power results for all segments, see 'Results in list mode'.

INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)

- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_maximum_max: Comma-separated list of power values, one value per measured segment

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[float] = driver.gprf.measurement.power.peak.maximum.read()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_maximum_max: Comma-separated list of power values, one value per measured segment

6.7.1.8.11.2 Minimum

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
READ:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
```

class MinimumCls

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

calculate() → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.peak.minimum.
↪calculate()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_minimum_min: Comma-separated list of power values, one value per measured segment

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[float] = driver.gprf.measurement.power.peak.minimum.fetch()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_minimum_min: Comma-separated list of power values, one value per measured segment

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[float] = driver.gprf.measurement.power.peak.minimum.read()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_minimum_min: Comma-separated list of power values, one value per measured segment

6.7.1.8.12 StandardDev

SCPI Commands :

```
CALCulate:GPRF:MEASurement<Instance>:POWer:SDEViation
FETCh:GPRF:MEASurement<Instance>:POWer:SDEViation
READ:GPRF:MEASurement<Instance>:POWer:SDEViation
```

class StandardDevCls

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

calculate() → List[ResultStatus2]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[enums.ResultStatus2] = driver.gprf.measurement.power.standardDev.
↪calculate()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)

- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_std_dev_cur: Comma-separated list of power values, one value per measured segment

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[float] = driver.gprf.measurement.power.standardDev.fetch()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)
- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_std_dev_cur: Comma-separated list of power values, one value per measured segment

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[float] = driver.gprf.measurement.power.standardDev.read()
```

Returns power results for all segments, see 'Results in list mode'.
INTRO_CMD_HELP: The following results can be retrieved:

- Current RMS (...:POWer:CURRent?)
- Current Min. (...:MINimum:CURRent?)
- Current Max. (...:MAXimum:CURRent?)
- Average RMS (...:AVERage?)
- Minimum (...:PEAK:MINimum?)

- Maximum (...:PEAK:MAXimum?)
- Standard Deviation (...:SDEViation?)

The values described below are returned by FETCh and READ commands. CALCulate commands return error codes instead, one value for each result listed below.

Suppressed linked return values: reliability

return
power_std_dev_cur: Comma-separated list of power values, one value per measured segment

Cloning the Group

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

Subgroups

6.7.1.8.12.1 Current

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:POWer:SDEViation:CURRent
READ:GPRF:MEASurement<Instance>:POWer:SDEViation:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:SDEViation:CURRent
value: List[float] = driver.gprf.measurement.power.standardDev.current.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power_std_dev_cur: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:SDEViation:CURRent
value: List[float] = driver.gprf.measurement.power.standardDev.current.read()
```

No command help available

Suppressed linked return values: reliability

return
power_std_dev_cur: No help available

6.7.1.8.13 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:STATe
value: enums.ResourceState = driver.gprf.measurement.power.state.fetch(timeout␣
↪= 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_state = enums.
↪TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

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

Subgroups

6.7.1.8.13.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:POWer:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.power.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.1.9 Spectrum

SCPI Commands :

```
INITiate:GPRF:MEASurement<Instance>:SPECtrum
STOP:GPRF:MEASurement<Instance>:SPECtrum
ABORt:GPRF:MEASurement<Instance>:SPECtrum
```

class SpectrumCls

Spectrum commands group definition. 48 total commands, 8 Subgroups, 3 group commands

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:SPECtrum
driver.gprf.measurement.spectrum.abort()
```

No command help available

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

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:SPECtrum
driver.gprf.measurement.spectrum.initiate()
```

No command help available

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

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:SPECtrum
driver.gprf.measurement.spectrum.stop()
```

No command help available

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.gprf.measurement.spectrum.clone()
```

Subgroups

6.7.1.9.1 Average

class AverageCls

Average commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.spectrum.average.clone()
```

Subgroups

6.7.1.9.1.1 Average

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:AVERage
READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:AVERage
value: List[float] = driver.gprf.measurement.spectrum.average.average.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:AVERage
value: List[float] = driver.gprf.measurement.spectrum.average.average.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.1.2 Current

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:CURRent
READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:CURRent
value: List[float] = driver.gprf.measurement.spectrum.average.current.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:CURRent
value: List[float] = driver.gprf.measurement.spectrum.average.current.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.1.3 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MAXimum
READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.average.maximum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.average.maximum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.1.4 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MINimum
READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MINimum
value: List[float] = driver.gprf.measurement.spectrum.average.minimum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MINimum
value: List[float] = driver.gprf.measurement.spectrum.average.minimum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.2 Marker<Marker>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.gprf.measurement.spectrum.marker.repcap_marker_get()
driver.gprf.measurement.spectrum.marker.repcap_marker_set(repcap.Marker.Nr1)
```

class MarkerCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.spectrum.marker.clone()
```

Subgroups

6.7.1.9.2.1 Npeak

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MARKer<MarkerNo>:NPEak
```

class NpeakCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Xvalue: float: No parameter help available
- Yvalue: float: No parameter help available

fetch(*detector: Detector*, *statistic: Statistic*, *marker=Marker.Default*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MARKer<MarkerNo>:NPEak
value: FetchStruct = driver.gprf.measurement.spectrum.marker.npeak.
↪fetch(detector = enums.Detector.AUTopeak, statistic = enums.Statistic.AVERage,
↪ marker = repcap.Marker.Default)
```

No command help available

param detector
No help available

param statistic
No help available

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

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

6.7.1.9.3 Maximum

class MaximumCls

Maximum commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.spectrum.maximum.clone()
```

Subgroups

6.7.1.9.3.1 Average

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:AVERage
READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:AVERage
value: List[float] = driver.gprf.measurement.spectrum.maximum.average.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:AVERage
value: List[float] = driver.gprf.measurement.spectrum.maximum.average.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.3.2 Current

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:CURRent
READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:CURRent
value: List[float] = driver.gprf.measurement.spectrum.maximum.current.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:CURRent
value: List[float] = driver.gprf.measurement.spectrum.maximum.current.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.3.3 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MAXimum
READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.maximum.maximum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.maximum.maximum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.3.4 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MINimum
READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MINimum
value: List[float] = driver.gprf.measurement.spectrum.maximum.minimum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MINimum
value: List[float] = driver.gprf.measurement.spectrum.maximum.minimum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.4 Minimum

class MinimumCls

Minimum commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.spectrum.minimum.clone()
```

Subgroups

6.7.1.9.4.1 Average

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:AVERage
READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:AVERage
value: List[float] = driver.gprf.measurement.spectrum.minimum.average.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:AVERage
value: List[float] = driver.gprf.measurement.spectrum.minimum.average.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.4.2 Current

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:CURRent
READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:CURRent
value: List[float] = driver.gprf.measurement.spectrum.minimum.current.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:CURRent
value: List[float] = driver.gprf.measurement.spectrum.minimum.current.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.4.3 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MAXimum
READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.minimum.maximum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.minimum.maximum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.4.4 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MINimum
READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MINimum
value: List[float] = driver.gprf.measurement.spectrum.minimum.minimum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MINimum
value: List[float] = driver.gprf.measurement.spectrum.minimum.minimum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.5 ReferenceMarker

class ReferenceMarkerCls

ReferenceMarker 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.gprf.measurement.spectrum.referenceMarker.clone()
```

Subgroups

6.7.1.9.5.1 Npeak

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:NPEak
```

class NpeakCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Xvalue: float: No parameter help available
- Yvalue: float: No parameter help available

fetch(*detector: Detector*, *statistic: Statistic*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:NPEak
value: FetchStruct = driver.gprf.measurement.spectrum.referenceMarker.npeak.
↪fetch(detector = enums.Detector.AUTopeak, statistic = enums.Statistic.AVERage)
```

No command help available

param detector
No help available

param statistic
No help available

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

6.7.1.9.5.2 Speak

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:SPEak
```

class SpeakCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Xvalue: float: No parameter help available
- Yvalue: float: No parameter help available

fetch(*detector: Detector*, *statistic: Statistic*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:SPEak
value: FetchStruct = driver.gprf.measurement.spectrum.referenceMarker.speak.
↪fetch(detector = enums.Detector.AUTopeak, statistic = enums.Statistic.AVERage)
```

No command help available

param detector
No help available

param statistic
No help available

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

6.7.1.9.6 Rms

class RmsCls

Rms commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.spectrum.rms.clone()
```

Subgroups

6.7.1.9.6.1 Average

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:AVERage
READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:AVERage
value: List[float] = driver.gprf.measurement.spectrum.rms.average.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:AVERage
value: List[float] = driver.gprf.measurement.spectrum.rms.average.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.6.2 Current

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:CURRent
READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:CURRent
value: List[float] = driver.gprf.measurement.spectrum.rms.current.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:CURRent
value: List[float] = driver.gprf.measurement.spectrum.rms.current.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.6.3 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:MAXimum
READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.rms.maximum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.rms.maximum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.6.4 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:MINimum
READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:RMS:MINimum
value: List[float] = driver.gprf.measurement.spectrum.rms.minimum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MINimum
value: List[float] = driver.gprf.measurement.spectrum.rms.minimum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.7 Sample

class SampleCls

Sample commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.spectrum.sample.clone()
```

Subgroups

6.7.1.9.7.1 Average

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:AVERage
READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:AVERage
value: List[float] = driver.gprf.measurement.spectrum.sample.average.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:AVERage
value: List[float] = driver.gprf.measurement.spectrum.sample.average.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.7.2 Current

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:CURRent
READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:CURRent
value: List[float] = driver.gprf.measurement.spectrum.sample.current.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:CURRent
value: List[float] = driver.gprf.measurement.spectrum.sample.current.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.7.3 Maximum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MAXimum
READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.sample.maximum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MAXimum
value: List[float] = driver.gprf.measurement.spectrum.sample.maximum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.7.4 Minimum

SCPI Commands :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MINimum
READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MINimum
value: List[float] = driver.gprf.measurement.spectrum.sample.minimum.fetch()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MINimum
value: List[float] = driver.gprf.measurement.spectrum.sample.minimum.read()
```

No command help available

Suppressed linked return values: reliability

return
power: No help available

6.7.1.9.8 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe
value: enums.ResourceState = driver.gprf.measurement.spectrum.state.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

No command help available

param timeout
No help available

param target_main_state
No help available

param target_sync_state
No help available

return
meas_state: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprf.measurement.spectrum.state.clone()
```

Subgroups

6.7.1.9.8.1 All

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe:ALL
value: List[enums.ResourceState] = driver.gprf.measurement.spectrum.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

No command help available

param timeout
No help available

param target_main_state
No help available

param target_sync_state
No help available

return
meas_state: No help available

6.8 Modify

class ModifyCls

Modify 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.modify.clone()
```

Subgroups

6.8.1 System

class SystemCls

System 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.modify.system.clone()
```

Subgroups

6.8.1.1 Attenuation

class AttenuationCls

Attenuation 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.modify.system.attenuation.clone()
```

Subgroups

6.8.1.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.modify.system.attenuation.correctionTable.clone()
```

Subgroups

6.8.1.1.1.1 Globale

SCPI Command :

```
MODify:SYSTem:ATTenuation:CTABle:GLOBal
```

class GlobaleCls

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

set(*name: str*, *frequency: List[float]*, *attenuation: List[float]*) → None

```
# SCPI: MODify:SYSTem:ATTenuation:CTABle:GLOBal
driver.modify.system.attenuation.correctionTable.globale.set(name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3], attenuation = [1.1, 2.2, 3.3])
```

No command help available

param name
No help available

param frequency
No help available

param attenuation
No help available

6.8.1.1.1.2 Tenvironment

SCPI Command :

```
MODify:SYSTem:ATTenuation:CTABle[:TENVironment]
```

class TenvironmentCls

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

set(*name: str*, *frequency: List[float]*, *attenuation: List[float]*) → None

```
# SCPI: MODify:SYSTem:ATTenuation:CTABle[:TENVironment]
driver.modify.system.attenuation.correctionTable.tenvironment.set(name = 'abc',␣
↪frequency = [1.1, 2.2, 3.3], attenuation = [1.1, 2.2, 3.3])
```

No command help available

param name
No help available

param frequency
No help available

param attenuation
No help available

6.9 Remove

class RemoveCls

Remove 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.remove.clone()
```

Subgroups

6.9.1 System

class SystemCls

System 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.remove.system.clone()
```

Subgroups

6.9.1.1 Attenuation

class AttenuationCls

Attenuation 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.remove.system.attenuation.clone()
```

Subgroups

6.9.1.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.remove.system.attenuation.correctionTable.clone()
```

Subgroups

6.9.1.1.1.1 Globale

SCPI Command :

```
REMove:SYSTem:ATTenuation:CTABle:GLOBal
```

class GlobaleCls

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

set(*name: str*, *intervall_start: float*, *intervall_end: float = None*) → None

```
# SCPI: REMove:SYSTem:ATTenuation:CTABle:GLOBal
driver.remove.system.attenuation.correctionTable.globale.set(name = 'abc',␣
↪intervall_start = 1.0, intervall_end = 1.0)
```

No command help available

param name
No help available

param intervall_start
No help available

param intervall_end
No help available

6.9.1.1.1.2 Tenvironment

SCPI Command :

```
REMove:SYSTem:ATTenuation:CTABle[:TENVironment]
```

class TenvironmentCls

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

set(*name: str*, *intervall_start: float*, *intervall_end: float = None*) → None

```
# SCPI: REMove:SYSTem:ATTenuation:CTABle[:TENVironment]
driver.remove.system.attenuation.correctionTable.tenvironment.set(name = 'abc',␣
↪intervall_start = 1.0, intervall_end = 1.0)
```

No command help available

param name
No help available

param intervall_start
No help available

param intervall_end
No help available

6.9.2 Tenvironment

class TenvironmentCls

Tenvironment 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.remove.tenvironment.clone()
```

Subgroups

6.9.2.1 Spath

class SpathCls

Spath 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.remove.tenvironment.spath.clone()
```

Subgroups

6.9.2.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable 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.remove.tenvironment.spath.correctionTable.clone()
```

Subgroups

6.9.2.1.1.1 Rx

SCPI Command :

```
REMove:TENVironment:SPATh:CTABle:RX
```

class RxCls

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

set(*name_signal_path: str*, *correction_table: List[str] = None*) → None

```
# SCPI: REMove:TENVironment:SPATh:CTABle:RX
driver.remove.tenvironment.spath.correctionTable.rx.set(name_signal_path = 'abc
↪', correction_table = ['abc1', 'abc2', 'abc3'])
```

No command help available

param name_signal_path
No help available

param correction_table
No help available

6.9.2.1.1.2 Tx

SCPI Command :

```
REMove:TENVironment:SPATh:CTABle:TX
```

class TxCls

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

set(*name_signal_path: str*, *correction_table: List[str] = None*) → None

```
# SCPI: REMove:TENVironment:SPATh:CTABle:TX
driver.remove.tenvironment.spath.correctionTable.tx.set(name_signal_path = 'abc
↪', correction_table = ['abc1', 'abc2', 'abc3'])
```

No command help available

param name_signal_path
No help available

param correction_table
No help available

6.10 Results

class ResultsCls

Results 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.results.clone()
```

Subgroups

6.10.1 Gprf

class GprfCls

Gprf 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.results.gprf.clone()
```

Subgroups

6.10.1.1 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.results.gprf.measurement.clone()
```

Subgroups

6.10.1.1.1 Power

class PowerCls

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

Cloning the Group

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

Subgroups

6.10.1.1.1.1 Current

SCPI Command :

```
FETCh:RESults:GPRF:MEASurement<Instance>:POWer:CURRent
```

class CurrentCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Stat_Count: int: No parameter help available
- Power_Current_Rms: List[float]: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:RESults:GPRF:MEASurement<Instance>:POWer:CURRent
value: FetchStruct = driver.results.gprf.measurement.power.current.fetch()
```

No command help available

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

6.11 Route

class RouteCls

Route commands group definition. 32 total commands, 14 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.11.1 Bluetooth

class BluetoothCls

Bluetooth 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.route.bluetooth.clone()
```

Subgroups

6.11.1.1 Measurement

class MeasurementCls

Measurement 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.route.bluetooth.measurement.clone()
```

Subgroups

6.11.1.1.1 Spath

SCPI Commands :

```
ROUTe:BLUetooth:MEASurement<Instance>:SPATh:COUNt
ROUTe:BLUetooth:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:BLUetooth:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.bluetooth.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:BLUetooth:MEASurement<Instance>:SPATh
value: List[str] = driver.route.bluetooth.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:BLUetooth:MEASurement<Instance>:SPATh
driver.route.bluetooth.measurement.spath.set_value(signal_path = ['abc1', 'abc2
↪', 'abc3'])
```

No command help available

param signal_path
No help available

6.11.2 Cdma

class CdmaCls

Cdma 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.route.cdma.clone()
```

Subgroups

6.11.2.1 Measurement

SCPI Command :

```
ROUTe:CDMA:MEASurement<Instance>:SPATh
```

class MeasurementCls

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

get_spath() → List[str]

```
# SCPI: ROUTe:CDMA:MEASurement<Instance>:SPATh
value: List[str] = driver.route.cdma.measurement.get_spath()
```

No command help available

return
signal_path: No help available

set_spath(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:CDMA:MEASurement<Instance>:SPATh
driver.route.cdma.measurement.set_spath(signal_path = ['abc1', 'abc2', 'abc3'])
```

No command help available

param signal_path
No help available

6.11.3 Gprf

class GprfCls

Gprf 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.route.gprf.clone()
```

Subgroups

6.11.3.1 Generator

class GeneratorCls

Generator 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.route.gprf.generator.clone()
```

Subgroups

6.11.3.1.1 RfSettings

SCPI Command :

```
ROUTe:GPRF:GENerator<Instance>:RFSettings:CONNector
```

class RfSettingsCls

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

get_connector() → TxConnector

```
# SCPI: ROUTe:GPRF:GENerator<Instance>:RFSettings:CONNector
value: enums.TxConnector = driver.route.gprf.generator.rfSettings.get_
↪connector()
```

No command help available

return
output_connector: No help available

set_connector(*output_connector: TxConnector*) → None

```
# SCPI: ROUTe:GPRF:GENerator<Instance>:RFSettings:CONNector
driver.route.gprf.generator.rfSettings.set_connector(output_connector = enums.
↪TxConnector.I120)
```

No command help available

param output_connector
No help available

6.11.3.1.2 Spath

SCPI Commands :

```
ROUTe:GPRF:GENerator<Instance>:SPATh:COUNt
ROUTe:GPRF:GENerator<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:GPRF:GENerator<Instance>:SPATh:COUNt
value: int = driver.route.gprf.generator.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → str

```
# SCPI: ROUTe:GPRF:GENerator<Instance>:SPATh
value: str = driver.route.gprf.generator.spath.get_value()
```

Selects the RF connection (broadcast off) or connector group (broadcast on) for the generated signal. For possible values, see method **RsCMPX_Gprf.Catalog.Gprf.Generator.Spath.get_**.

return
signal_path: No help available

set_value(*signal_path: str*) → None

```
# SCPI: ROUTe:GPRF:GENerator<Instance>:SPATh
driver.route.gprf.generator.spath.set_value(signal_path = 'abc')
```

Selects the RF connection (broadcast off) or connector group (broadcast on) for the generated signal. For possible values, see method **RsCMPX_Gprf.Catalog.Gprf.Generator.Spath.get_**.

param signal_path
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.gprf.generator.spath.clone()
```

Subgroups

6.11.3.1.2.1 Group

SCPI Command :

```
ROUTe:GPRF:GENerator<Instance>:SPATh:GROup
```

class GroupCls

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

get(*connector_name: str*) → str

```
# SCPI: ROUTe:GPRF:GENerator<Instance>:SPATh:GROup
value: str = driver.route.gprf.generator.spath.group.get(connector_name = 'abc')
```

Assigns an RF connection to an output connector. The connection is used if you activate the connector for signal output via a connector group. This command is only relevant if you create your own RF connections (see base manual) . The default RF connections have the same name as the assigned connector.

param connector_name
Name of the output connector.

return
signal_path: RF connection to be assigned to the connector.

set(*connector_name: str*, *signal_path: str*) → None

```
# SCPI: ROUTe:GPRF:GENerator<Instance>:SPATh:GROup
driver.route.gprf.generator.spath.group.set(connector_name = 'abc', signal_path␣
↪= 'abc')
```

Assigns an RF connection to an output connector. The connection is used if you activate the connector for signal output via a connector group. This command is only relevant if you create your own RF connections (see base manual) . The default RF connections have the same name as the assigned connector.

param connector_name
Name of the output connector.

param signal_path
RF connection to be assigned to the connector.

6.11.3.2 Measurement

class MeasurementCls

Measurement commands group definition. 4 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.11.3.2.1 RfSettings

SCPI Command :

```
ROUTe:GPRF:MEASurement<Instance>:RFSettings:CONNector
```

class RfSettingsCls

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

get_connector() → RfConnector

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:RFSettings:CONNector
value: enums.RfConnector = driver.route.gprf.measurement.rfSettings.get_
↪connector()
```

No command help available

return
rf_input_con: No help available

set_connector(*rf_input_con: RfConnector*) → None

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:RFSettings:CONNector
driver.route.gprf.measurement.rfSettings.set_connector(rf_input_con = enums.
↪RfConnector.I11I)
```

No command help available

param rf_input_con
No help available

6.11.3.2.2 Scenario

class ScenarioCls

Scenario 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.route.gprf.measurement.scenario.clone()
```

Subgroups

6.11.3.2.2.1 Catalog

SCPI Command :

```
ROUTe:GPRF:MEASurement<Instance>:SCENario:CATalog:CSPath
```

class CatalogCls

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

get_cspath() → List[str]

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:CATalog:CSPath
value: List[str] = driver.route.gprf.measurement.scenario.catalog.get_cspath()
```

No command help available

return
csp_masters: No help available

6.11.3.2.3 Spath

SCPI Commands :

```
ROUTe:GPRF:MEASurement<Instance>:SPATh:COUNt
ROUTe:GPRF:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.gprf.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → str

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SPATh
value: str = driver.route.gprf.measurement.spath.get_value()
```

Selects one or more RF connections (signal input paths) for the measured signal. The number of expected connections depends on the list mode settings of the power measurement. Configure them before sending this command.

INTRO_CMD_HELP: Distinguish the following situations:

- List mode OFF: One connection is expected.
- List mode ON and connection source GLOBal: One connection is expected. It is used for all list mode segments.
- List mode ON and connection source INDex: The number of connections configured via [CONFigure:]GPRF:MEAS<i>:POWer:LIST:NIDX is expected. The order of the connections assigns them to an index (connection with index 1, index 2, index 3, ...) .

INTRO_CMD_HELP: Related commands:

- List mode state: method RsCMPX_Gprf.Configure.Gprf.Measurement.Power.ListPy.value
- Connection source: [CONFigure:]GPRF:MEAS<i>:POWer:LIST:CSOurce
- Connection index per segment: [CONFigure:]GPRF:MEAS<i>:POWer:LIST:CIDX

For possible connection names, see method **RsCMPX_Gprf.Catalog.Gprf.Measurement.Spath.get_**.

return
signal_path: Comma-separated list of strings, one string per RF connection.

set_value(*signal_path: str*) → None

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SPATh
driver.route.gprf.measurement.spath.set_value(signal_path = 'abc')
```

Selects one or more RF connections (signal input paths) for the measured signal. The number of expected connections depends on the list mode settings of the power measurement. Configure them before sending this command.

INTRO_CMD_HELP: Distinguish the following situations:

- List mode OFF: One connection is expected.
- List mode ON and connection source GLOBal: One connection is expected. It is used for all list mode segments.
- List mode ON and connection source INDex: The number of connections configured via [CONFigure:]GPRF:MEAS<i>:POWer:LIST:NIDX is expected. The order of the connections assigns them to an index (connection with index 1, index 2, index 3, ...) .

INTRO_CMD_HELP: Related commands:

- List mode state: method RsCMPX_Gprf.Configure.Gprf.Measurement.Power.ListPy.value
- Connection source: [CONFigure:]GPRF:MEAS<i>:POWer:LIST:CSOurce
- Connection index per segment: [CONFigure:]GPRF:MEAS<i>:POWer:LIST:CIDX

For possible connection names, see method **RsCMPX_Gprf.Catalog.Gprf.Measurement.Spath.get_**.

param signal_path
Comma-separated list of strings, one string per RF connection.

6.11.4 Gsm

class GsmCls

Gsm 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.route.gsm.clone()
```

Subgroups

6.11.4.1 Measurement

class MeasurementCls

Measurement 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.route.gsm.measurement.clone()
```

Subgroups

6.11.4.1.1 Spath

SCPI Commands :

```
ROUTe:GSM:MEASurement<Instance>:SPATh:COUNt
ROUTe:GSM:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:GSM:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.gsm.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:GSM:MEASurement<Instance>:SPATh
value: List[str] = driver.route.gsm.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:GSM:MEASurement<Instance>:SPATh
driver.route.gsm.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.5 Lte

class LteCls

Lte 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.route.lte.clone()
```

Subgroups

6.11.5.1 Measurement

class MeasurementCls

Measurement 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.route.lte.measurement.clone()
```

Subgroups

6.11.5.1.1 Spath

SCPI Commands :

```
ROUTe:LTE:MEASurement<Instance>:SPATh:COUNt
ROUTe:LTE:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:LTE:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.lte.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:LTE:MEASurement<Instance>:SPATh
value: List[str] = driver.route.lte.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:LTE:MEASurement<Instance>:SPATh
driver.route.lte.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.6 LteDl

class LteDlCls

LteDl 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.route.lteDl.clone()
```

Subgroups

6.11.6.1 Measurement

class MeasurementCls

Measurement 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.route.lteDl.measurement.clone()
```

Subgroups

6.11.6.1.1 Spath

SCPI Commands :

```
ROUTe:LTEDl:MEASurement<Instance>:SPATh:COUNt
ROUTe:LTEDl:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:LTEDl:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.lteDl.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:LTEDl:MEASurement<Instance>:SPATh
value: List[str] = driver.route.lteDl.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:LTEDl:MEASurement<Instance>:SPATh
driver.route.lteDl.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.7 Niot

class NiotCls

Niot 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.route.niot.clone()
```

Subgroups

6.11.7.1 Measurement

SCPI Command :

```
ROUTe:NIOT:MEASurement<Instance>:SPATh
```

class MeasurementCls

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

get_spath() → List[str]

```
# SCPI: ROUTe:NIOT:MEASurement<Instance>:SPATh
value: List[str] = driver.route.niot.measurement.get_spath()
```

No command help available

return
signal_path: No help available

set_spath(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:NIOT:MEASurement<Instance>:SPATh
driver.route.niot.measurement.set_spath(signal_path = ['abc1', 'abc2', 'abc3'])
```

No command help available

param signal_path
No help available

6.11.8 NrDl

class NrDlCls

NrDl 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.route.nrDl.clone()
```

Subgroups

6.11.8.1 Measurement

class MeasurementCls

Measurement 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.route.nrDl.measurement.clone()
```

Subgroups

6.11.8.1.1 Spath

SCPI Commands :

```
ROUTe:NRDL:MEASurement<Instance>:SPATh:COUNt
ROUTe:NRDL:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:NRDL:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.nrDl.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:NRDL:MEASurement<Instance>:SPATh
value: List[str] = driver.route.nrDl.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:NRDL:MEASurement<Instance>:SPATh
driver.route.nrDl.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.9 NrMmw

class NrMmwCls

NrMmw 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.route.nrMmw.clone()
```

Subgroups

6.11.9.1 Measurement

class MeasurementCls

Measurement 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.route.nrMmw.measurement.clone()
```

Subgroups

6.11.9.1.1 Spath

SCPI Commands :

```
ROUTe:NRMMw:MEASurement<Instance>:SPATh:COUNt
ROUTe:NRMMw:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:NRMMw:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.nrMmw.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:NRMMw:MEASurement<Instance>:SPATh
value: List[str] = driver.route.nrMmw.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:NRMMw:MEASurement<Instance>:SPATh
driver.route.nrMmw.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.10 NrSub

class NrSubCls

NrSub 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.route.nrSub.clone()
```

Subgroups

6.11.10.1 Measurement

class MeasurementCls

Measurement 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.route.nrSub.measurement.clone()
```

Subgroups

6.11.10.1.1 Spath

SCPI Commands :

```
ROUTe:NRSub:MEASurement<Instance>:SPATh:COUNt
ROUTe:NRSub:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:NRSub:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.nrSub.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:NRSub:MEASurement<Instance>:SPATh
value: List[str] = driver.route.nrSub.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:NRSub:MEASurement<Instance>:SPATh
driver.route.nrSub.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.11 Uwb

class UwbCls

Uwb 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.route.uwb.clone()
```

Subgroups

6.11.11.1 Measurement

class MeasurementCls

Measurement 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.route.uwb.measurement.clone()
```

Subgroups

6.11.11.1.1 Spath

SCPI Commands :

```
ROUTe:UWB:MEASurement<Instance>:SPATh:COUNt
ROUTe:UWB:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:UWB:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.uwb.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:UWB:MEASurement<Instance>:SPATh
value: List[str] = driver.route.uwb.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:UWB:MEASurement<Instance>:SPATh
driver.route.uwb.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.12 Wcdma

class WcdmaCls

Wcdma 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.route.wcdma.clone()
```

Subgroups

6.11.12.1 Measurement

class MeasurementCls

Measurement 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.route.wcdma.measurement.clone()
```

Subgroups

6.11.12.1.1 Spath

SCPI Commands :

```
ROUTe:WCDMa:MEASurement<Instance>:SPATh:COUNt
ROUTe:WCDMa:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:WCDMa:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.wcdma.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:WCDMa:MEASurement<Instance>:SPATh
value: List[str] = driver.route.wcdma.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:WCDMa:MEASurement<Instance>:SPATh
driver.route.wcdma.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.13 Wlan

class WlanCls

Wlan 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.route.wlan.clone()
```

Subgroups

6.11.13.1 Measurement

class MeasurementCls

Measurement 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.route.wlan.measurement.clone()
```

Subgroups

6.11.13.1.1 Spath

SCPI Commands :

```
ROUTe:WLAN:MEASurement<Instance>:SPATh:COUNt
ROUTe:WLAN:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:WLAN:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.wlan.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:WLAN:MEASurement<Instance>:SPATh
value: List[str] = driver.route.wlan.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:WLAN:MEASurement<Instance>:SPATh
driver.route.wlan.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.11.14 Wpan

class WpanCls

Wpan 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.route.wpan.clone()
```

Subgroups

6.11.14.1 Measurement

class MeasurementCls

Measurement 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.route.wpan.measurement.clone()
```

Subgroups

6.11.14.1.1 Spath

SCPI Commands :

```
ROUTe:WPAN:MEASurement<Instance>:SPATh:COUNt
ROUTe:WPAN:MEASurement<Instance>:SPATh
```

class SpathCls

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

get_count() → int

```
# SCPI: ROUTe:WPAN:MEASurement<Instance>:SPATh:COUNt
value: int = driver.route.wpan.measurement.spath.get_count()
```

No command help available

return
signal_path_count: No help available

get_value() → List[str]

```
# SCPI: ROUTe:WPAN:MEASurement<Instance>:SPATh
value: List[str] = driver.route.wpan.measurement.spath.get_value()
```

No command help available

return
signal_path: No help available

set_value(*signal_path: List[str]*) → None

```
# SCPI: ROUTe:WPAN:MEASurement<Instance>:SPATh
driver.route.wpan.measurement.spath.set_value(signal_path = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

param signal_path
No help available

6.12 Sense

class SenseCls

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

Cloning the Group

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

Subgroups

6.12.1 Base

class BaseCls

Base 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.base.clone()
```

Subgroups

6.12.1.1 Temperature

class TemperatureCls

Temperature 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.base.temperature.clone()
```

Subgroups

6.12.1.1.1 Operating

class OperatingCls

Operating 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.base.temperature.operating.clone()
```

Subgroups

6.12.1.1.1.1 Ambient

SCPI Command :

```
SENSe:BASE:TEMPerature:OPERating:AMBient
```

class AmbientCls

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

class GetStruct

Response structure. Fields:

- Temperature: float: No parameter help available
- Timestamp: str: No parameter help available
- Box: str: No parameter help available

get(*all_py: All = None*) → GetStruct

```
# SCPI: SENSe:BASE:TEMPerature:OPERating:AMBient
value: GetStruct = driver.sense.base.temperature.operating.ambient.get(all_py =␣
↪enums.All.ALL)
```

No command help available

param all_py
No help available

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

6.12.2 System

class SystemCls

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

Cloning the Group

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

Subgroups

6.12.2.1 Positioner<Positioner>

RepCap Settings

```
# Range: Ix1 .. Ix32
rc = driver.sense.system.positioner.repcap_positioner_get()
driver.sense.system.positioner.repcap_positioner_set(repcap.Positioner.Ix1)
```

class PositionerCls

Positioner commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Positioner, default value after init: Positioner.Ix1

Cloning the Group

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

Subgroups

6.12.2.1.1 IsMoving

SCPI Command :

```
SENSe:SYSTem:POSitioner<PositionerIdx>:ISMoving
```

class IsMovingCls

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

get(*positioner=Positioner.Default*) → bool

```
# SCPI: SENSe:SYSTem:POSitioner<PositionerIdx>:ISMoving
value: bool = driver.sense.system.positioner.isMoving.get(positioner = repcap.
↪Positioner.Default)
```

No command help available

param positioner
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Positioner')

return
is_moving: No help available

6.12.2.1.2 Position

SCPI Command :

```
SENSe:SYSTem:POSitioner<PositionerIdx>:POSition
```

class PositionCls

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

class GetStruct

Response structure. Fields:

- Theta: float: No parameter help available
- Phi: float: No parameter help available

get(*positioner=Positioner.Default*) → GetStruct

```
# SCPI: SENSe:SYSTem:POSitioner<PositionerIdx>:POSition
value: GetStruct = driver.sense.system.positioner.position.get(positioner =␣
↪repcap.Positioner.Default)
```

No command help available

param positioner
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Positioner')

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

6.13 Source

class SourceCls

Source commands group definition. 186 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

6.13.1 Gprf

class GprfCls

Gprf commands group definition. 186 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.13.1.1 Generator

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:BBMode
```

class GeneratorCls

Generator commands group definition. 186 total commands, 8 Subgroups, 1 group commands

get_bb_mode() → BasebandMode

```
# SCPI: SOURce:GPRF:GENerator<Instance>:BBMode
value: enums.BasebandMode = driver.source.gprf.generator.get_bb_mode()
```

Selects the baseband mode for the generator signal.

return
baseband_mode: CW: unmodulated continuous wave signal DTONe: dual-tone signal ARB: processing a waveform file

set_bb_mode(*baseband_mode: BasebandMode*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:BBMode
driver.source.gprf.generator.set_bb_mode(baseband_mode = enums.BasebandMode.ARB)
```

Selects the baseband mode for the generator signal.

param baseband_mode
CW: unmodulated continuous wave signal DTONe: dual-tone signal ARB: processing a waveform file

Cloning the Group

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

Subgroups

6.13.1.1.1 Arb

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:ARB:FOFFset
SOURce:GPRF:GENerator<Instance>:ARB:SCOunt
SOURce:GPRF:GENerator<Instance>:ARB:ASAMples
SOURce:GPRF:GENerator<Instance>:ARB:REPetition
SOURce:GPRF:GENerator<Instance>:ARB:CYCLes
SOURce:GPRF:GENerator<Instance>:ARB:POFFset
SOURce:GPRF:GENerator<Instance>:ARB:CRATe
SOURce:GPRF:GENerator<Instance>:ARB:LOFFset
SOURce:GPRF:GENerator<Instance>:ARB:CRCProtect
SOURce:GPRF:GENerator<Instance>:ARB:STATus
```

class ArbCls

Arb commands group definition. 28 total commands, 6 Subgroups, 10 group commands

class ScountStruct

Structure for reading output parameters. Fields:

- Sample_Count_Time: float: No parameter help available
- Sample_Count: List[int]: No parameter help available

get_asamples() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:ASAMples
value: int = driver.source.gprf.generator.arb.get_asamples()
```

No command help available

return
add_samples: No help available

get_crate() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:CRATe
value: float = driver.source.gprf.generator.arb.get_crate()
```

Queries the clock rate of the loaded waveform file.

return
clock_rate: No help available

get_crc_protect() → YesNoStatus

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:CRCProtect
value: enums.YesNoStatus = driver.source.gprf.generator.arb.get_crc_protect()
```

Indicates whether the loaded ARB file contains a CRC checksum. To get a valid result, the related ARB file must be loaded into the memory. That means, the baseband mode must be ARB and the generator state must be ON. Otherwise, NAV is returned.

return
crc_protection: No help available

get_cycles() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:CYCLes
value: int = driver.source.gprf.generator.arb.get_cycles()
```

No command help available

return
cycles: No help available

get_foffset() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:FOFFset
value: float = driver.source.gprf.generator.arb.get_foffset()
```

No command help available

return
frequency_offset: No help available

get_loffset() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:LOFFset
value: float = driver.source.gprf.generator.arb.get_loffset()
```

Queries the level offset (peak to average ratio, PAR) of the loaded waveform file. The PAR is equal to the absolute value of the difference between the RMS Offset and the Peak Offset (crest factor) .

return
level_offset: No help available

get_poffset() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:POFFset
value: float = driver.source.gprf.generator.arb.get_poffset()
```

Queries the peak offset of the loaded waveform file.

return
peak_offset: No help available

get_repetition() → RepeatMode

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:REPetition
value: enums.RepeatMode = driver.source.gprf.generator.arb.get_repetition()
```

No command help available

return
repetition: No help available

get_scount() → ScountStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:SCOunt
value: ScountStruct = driver.source.gprf.generator.arb.get_scount()
```

No command help available

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

get_status() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:STATus
value: int = driver.source.gprf.generator.arb.get_status()
```

No command help available

return
arb_segment_no: No help available

set_asamples(*add_samples: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:ASAMples
driver.source.gprf.generator.arb.set_asamples(add_samples = 1)
```

No command help available

param add_samples
No help available

set_cycles(*cycles: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:CYCLes
driver.source.gprf.generator.arb.set_cycles(cycles = 1)
```

No command help available

param cycles
No help available

set_foffset(*frequency_offset: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:FOFFset
driver.source.gprf.generator.arb.set_foffset(frequency_offset = 1.0)
```

No command help available

param frequency_offset
No help available

set_repetition(*repetition: RepeatMode*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:REPetition
driver.source.gprf.generator.arb.set_repetition(repetition = enums.RepeatMode.
↪CONTinuous)
```

No command help available

param repetition
No help available

Cloning the Group

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

Subgroups

6.13.1.1.1.1 File

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:ARB:FILE
SOURce:GPRF:GENerator<Instance>:ARB:FILE:DATE
SOURce:GPRF:GENerator<Instance>:ARB:FILE:VERSion
SOURce:GPRF:GENerator<Instance>:ARB:FILE:OPTion
```

class FileCls

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

get(*arb_file: ArbFile = None*) → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:FILE
value: str = driver.source.gprf.generator.arb.file.get(arb_file = enums.ArbFile.
↪ABSPath)
```

Selects a waveform file for the ARB baseband mode. This command supports path aliases (e.g. @WAVEFORM) . Use MMEMory:ALIases? to query the available path aliases. If the selected file does not exist or no file has been selected, a query returns 'No File Selected'.

INTRO_CMD_HELP: If the selected file does exist, a query returns:

- Without <PathType>: The string used to select the file. If an alias has been used, the alias is not substituted.
- With <PathType>: The absolute path of the file. If an alias has been used, the alias is substituted.

param arb_file
(enum or string) Name of the waveform file to be used (.wv) .

return
arb_file_retrun: No help available

get_date() → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:FILE:DATE
value: str = driver.source.gprf.generator.arb.file.get_date()
```

Queries the date of the loaded waveform file.

return
date: No help available

get_option() → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:FILE:OPTion
value: str = driver.source.gprf.generator.arb.file.get_option()
```

Returns the options that are required to play the loaded ARB file.

return
options: A comma-separated list of options.

get_version() → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:FILE:VERSion
value: str = driver.source.gprf.generator.arb.file.get_version()
```

Queries the version of the loaded waveform file.

return
version: The version or an empty string, if no file version is defined.

set(*arb_file: ArbFile = None*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:FILE
driver.source.gprf.generator.arb.file.set(arb_file = enums.ArbFile.ABSPath)
```

Selects a waveform file for the ARB baseband mode. This command supports path aliases (e.g. @WAVEFORM) . Use MMEMory:ALIases? to query the available path aliases. If the selected file does not exist or no file has been selected, a query returns 'No File Selected'.

INTRO_CMD_HELP: If the selected file does exist, a query returns:

- Without <PathType>: The string used to select the file. If an alias has been used, the alias is not substituted.
- With <PathType>: The absolute path of the file. If an alias has been used, the alias is substituted.

param arb_file
(enum or string) Name of the waveform file to be used (.wv) .

6.13.1.1.1.2 Marker

class MarkerCls

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

Cloning the Group

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

Subgroups

6.13.1.1.1.3 Delays

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:ARB:MARKer:DELays
```

class DelaysCls

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

class DelaysStruct

Response structure. Fields:

- Marker_2: int: No parameter help available
- Marker_3: int: No parameter help available
- Marker_4: int: No parameter help available
- Restart_Marker: int: No parameter help available

get() → DelaysStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MARKer:DELays
value: DelaysStruct = driver.source.gprf.generator.arb.marker.delays.get()
```

No command help available

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

set(*marker_2: int*, *marker_3: int*, *marker_4: int*, *restart_marker: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MARKer:DELays
driver.source.gprf.generator.arb.marker.delays.set(marker_2 = 1, marker_3 = 1,␣
↪marker_4 = 1, restart_marker = 1)
```

No command help available

param marker_2
No help available

param marker_3
No help available

param marker_4
No help available

param restart_marker
No help available

6.13.1.1.1.4 Msegment

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:NAME
SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:POFFset
SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:PAR
SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:DURation
SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:SAMPles
SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:CRATe
SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:NUMBer
```

class MsegmentCls

Msegment commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_crate() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:CRATe
value: List[float] = driver.source.gprf.generator.arb.msegment.get_crate()
```

No command help available

return
clock_rate: No help available

get_duration() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:DURation
value: List[float] = driver.source.gprf.generator.arb.msegment.get_duration()
```

No command help available

return
duration: No help available

get_name() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:NAME
value: List[str] = driver.source.gprf.generator.arb.msegment.get_name()
```

No command help available

return
name: No help available

get_number() → List[int]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:NUMBer
value: List[int] = driver.source.gprf.generator.arb.msegment.get_number()
```

No command help available

return
seg_number: No help available

get_par() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:PAR
value: List[float] = driver.source.gprf.generator.arb.msegment.get_par()
```

No command help available

return
par: No help available

get_poffset() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:POFFset
value: List[float] = driver.source.gprf.generator.arb.msegment.get_poffset()
```

No command help available

return
peak_offset: No help available

get_samples() → List[int]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:MSEGment:SAMPles
value: List[int] = driver.source.gprf.generator.arb.msegment.get_samples()
```

No command help available

return
samples: No help available

6.13.1.1.1.5 Samples

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:ARB:SAMPles
```

class SamplesCls

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

get_value() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:SAMPles
value: float = driver.source.gprf.generator.arb.samples.get_value()
```

Queries the number of samples in the loaded waveform file. The CMX500 supports waveform files with a size up to 512 MB.

return
samples: No help available

Cloning the Group

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

Subgroups

6.13.1.1.1.6 Range

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:ARB:SAMPles:RANGe
```

class RangeCls

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

class RangeStruct

Response structure. Fields:

- Range_Py: enums.Range: No parameter help available
- Start: int: No parameter help available
- Stop: int: No parameter help available

get() → RangeStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:SAMPles:RANGe
value: RangeStruct = driver.source.gprf.generator.arb.samples.range.get()
```

No command help available

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

set(*range_py: Range*, *start: int = None*, *stop: int = None*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:SAMPles:RANGe
driver.source.gprf.generator.arb.samples.range.set(range_py = enums.Range.FULL,␣
↪start = 1, stop = 1)
```

No command help available

param range_py
No help available

param start
No help available

param stop
No help available

6.13.1.1.1.7 Segments

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:ARB:SEGMents:NEXT
SOURce:GPRF:GENerator<Instance>:ARB:SEGMents:CURRent
```

class SegmentsCls

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

class CurrentStruct

Structure for reading output parameters. Fields:

- Segment_Number: int: No parameter help available
- Segment_Name: str: No parameter help available

get_current() → CurrentStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:SEGMents:CURRent
value: CurrentStruct = driver.source.gprf.generator.arb.segments.get_current()
```

No command help available

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

get_next() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:SEGMents:NEXT
value: int = driver.source.gprf.generator.arb.segments.get_next()
```

No command help available

return
segment_number: No help available

set_next(*segment_number: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:SEGMents:NEXT
driver.source.gprf.generator.arb.segments.set_next(segment_number = 1)
```

No command help available

param segment_number
No help available

6.13.1.1.1.8 UdMarker

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:ARB:UDMarker
```

class UdMarkerCls

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

class UdMarkerStruct

Response structure. Fields:

- Period: int: No parameter help available
- Start_State: enums.SignalSlope: No parameter help available
- Positions: List[int or bool]: No parameter help available

get() → UdMarkerStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:UDMarker
value: UdMarkerStruct = driver.source.gprf.generator.arb.udMarker.get()
```

No command help available

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

set(*period: int*, *start_state: SignalSlope*, *positions: List[int]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:UDMarker
driver.source.gprf.generator.arb.udMarker.set(period = 1, start_state = enums.
↪SignalSlope.FEDGe, positions = [1, True, 2, False, 3])
```

No command help available

param period
No help available

param start_state
No help available

param positions
(integer or boolean items) No help available

Cloning the Group

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

Subgroups

6.13.1.1.1.9 Clist

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:ARB:UDMarker:CLISt
```

class ClistCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:UDMarker:CLISt
driver.source.gprf.generator.arb.udMarker.clist.set()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:ARB:UDMarker:CLISt
driver.source.gprf.generator.arb.udMarker.clist.set_with_opc()
```

No command help available

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

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

6.13.1.1.2 Dtone

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:DTONe:RATio
```

class DtoneCls

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

get_ratio() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:DTONe:RATio
value: float = driver.source.gprf.generator.dtone.get_ratio()
```

Specifies the ratio in dB between the RMS levels of the two signals.

return
ratio: No help available

set_ratio(*ratio: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:DTONe:RATio
driver.source.gprf.generator.dtone.set_ratio(ratio = 1.0)
```

Specifies the ratio in dB between the RMS levels of the two signals.

param ratio
No help available

Cloning the Group

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

Subgroups

6.13.1.1.2.1 Level<LevelSource>

RepCap Settings

```
# Range: Src1 .. Src2
rc = driver.source.gprf.generator.dtone.level.repcap_levelSource_get()
driver.source.gprf.generator.dtone.level.repcap_levelSource_set(repcap.LevelSource.Src1)
```

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:DTONe:LEVel<source>
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: LevelSource, default value after init: LevelSource.Src1

get(*levelSource=LevelSource.Default*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:DTONe:LEVel<source>
value: float = driver.source.gprf.generator.dtone.level.get(levelSource =␣
↪repcap.LevelSource.Default)
```

Queries the output level of a source signal. The output level is a function of the generator output level and the ratio, see method RsCMPX_Gprf.Source.Gprf.Generator.RfSettings.level and method RsCMPX_Gprf.Source.Gprf.Generator.Dtone.ratio.

param levelSource
optional repeated capability selector. Default value: Src1 (settable in the interface 'Level')

return
level: No help available

Cloning the Group

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

6.13.1.1.2.2 Ofrequency<FrequencySource>

RepCap Settings

```
# Range: Src1 .. Src2
rc = driver.source.gprf.generator.dtone.ofrequency.repcap_frequencySource_get()
driver.source.gprf.generator.dtone.ofrequency.repcap_frequencySource_set(repcap.
↪FrequencySource.Src1)
```

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:DTONe:OFRequency<source>
```

class OfrequencyCls

Ofrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: FrequencySource, default value after init: FrequencySource.Src1

get(*frequencySource=FrequencySource.Default*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:DTONe:OFRequency<source>
value: float = driver.source.gprf.generator.dtone.ofrequency.
↪get(frequencySource = repcap.FrequencySource.Default)
```

Selects an offset frequency. The frequency of the modulated signal is equal to the base frequency (see method RsCMPX_Gprf. Source.Gprf.Generator.RfSettings.frequency) plus the offset frequency.

param frequencySource
optional repeated capability selector. Default value: Src1 (settable in the interface 'Ofrequency')

return
frequency: No help available

set(*frequency: float*, *frequencySource=FrequencySource.Default*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:DTONe:OFRequency<source>
driver.source.gprf.generator.dtone.ofrequency.set(frequency = 1.0,␣
↪frequencySource = repcap.FrequencySource.Default)
```

Selects an offset frequency. The frequency of the modulated signal is equal to the base frequency (see method RsCMPX_Gprf. Source.Gprf.Generator.RfSettings.frequency) plus the offset frequency.

param frequency
No help available

param frequencySource
optional repeated capability selector. Default value: Src1 (settable in the interface 'Ofrequency')

Cloning the Group

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

6.13.1.1.3 IqSettings

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:IQSettings:SRATe
SOURce:GPRF:GENerator<Instance>:IQSettings:TMODe
SOURce:GPRF:GENerator<Instance>:IQSettings:LEVel
SOURce:GPRF:GENerator<Instance>:IQSettings:PEP
SOURce:GPRF:GENerator<Instance>:IQSettings:CRESt
```

class IqSettingsCls

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

get_crest() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:IQSettings:CRESt
value: float = driver.source.gprf.generator.iqSettings.get_crest()
```

No command help available

return
crest: No help available

get_level() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:IQSettings:LEVel
value: float = driver.source.gprf.generator.iqSettings.get_level()
```

No command help available

return
level: No help available

get_pep() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:IQSettings:PEP
value: float = driver.source.gprf.generator.iqSettings.get_pep()
```

No command help available

return
pep: No help available

get_symbol_rate() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:IQSettings:SRATe
value: float = driver.source.gprf.generator.iqSettings.get_symbol_rate()
```

No command help available

return
sample_rate: No help available

get_tmode() → TransferMode

```
# SCPI: SOURce:GPRF:GENerator<Instance>:IQSettings:TMODe
value: enums.TransferMode = driver.source.gprf.generator.iqSettings.get_tmode()
```

No command help available

return
transfer_mode: No help available

set_symbol_rate(*sample_rate: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:IQSettings:SRATe
driver.source.gprf.generator.iqSettings.set_symbol_rate(sample_rate = 1.0)
```

No command help available

param sample_rate
No help available

set_tmode(*transfer_mode: TransferMode*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:IQSettings:TMODe
driver.source.gprf.generator.iqSettings.set_tmode(transfer_mode = enums.
↪TransferMode.ENABlemode)
```

No command help available

param transfer_mode
No help available

6.13.1.1.4 ListPy

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:AINDex
SOURce:GPRF:GENerator<Instance>:LIST:GOTO
SOURce:GPRF:GENerator<Instance>:LIST:REPetition
SOURce:GPRF:GENerator<Instance>:LIST:STARt
SOURce:GPRF:GENerator<Instance>:LIST:STOP
SOURce:GPRF:GENerator<Instance>:LIST:COUNt
SOURce:GPRF:GENerator<Instance>:LIST:MODE
SOURce:GPRF:GENerator<Instance>:LIST:CINDex
SOURce:GPRF:GENerator<Instance>:LIST
```

class ListPyCls

ListPy commands group definition. 33 total commands, 13 Subgroups, 9 group commands

get_aindex() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:AINDex
value: int = driver.source.gprf.generator.listPy.get_aindex()
```

No command help available

return
active_index: No help available

get_cindex() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:CINDex
value: int = driver.source.gprf.generator.listPy.get_cindex()
```

No command help available

return
current_index: No help available

get_count() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:COUNt
value: int = driver.source.gprf.generator.listPy.get_count()
```

No command help available

return
list_count: No help available

get_goto() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:GOTO
value: int = driver.source.gprf.generator.listPy.get_goto()
```

No command help available

return
goto_index: No help available

get_mode() → ListSubMode

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:MODE
value: enums.ListSubMode = driver.source.gprf.generator.listPy.get_mode()
```

No command help available

return
list_sub_mode: No help available

get_repetition() → RepeatMode

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:REPetition
value: enums.RepeatMode = driver.source.gprf.generator.listPy.get_repetition()
```

No command help available

return
repetition: No help available

get_start() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:STARt
value: int = driver.source.gprf.generator.listPy.get_start()
```

No command help available

return
start_index: No help available

get_stop() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:STOP
value: int = driver.source.gprf.generator.listPy.get_stop()
```

No command help available

return
stop_index: No help available

get_value() → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST
value: bool = driver.source.gprf.generator.listPy.get_value()
```

No command help available

return
enable_list_mode: No help available

set_cindex(*current_index: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:CINDex
driver.source.gprf.generator.listPy.set_cindex(current_index = 1)
```

No command help available

param current_index
No help available

set_goto(*goto_index: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:GOTO
driver.source.gprf.generator.listPy.set_goto(goto_index = 1)
```

No command help available

param goto_index
No help available

set_mode(*list_sub_mode: ListSubMode*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:MODE
driver.source.gprf.generator.listPy.set_mode(list_sub_mode = enums.ListSubMode.
↪AUTO)
```

No command help available

param list_sub_mode
No help available

set_repetition(*repetition: RepeatMode*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:REPetition
driver.source.gprf.generator.listPy.set_repetition(repetition = enums.
↪RepeatMode.CONTinuous)
```

No command help available

param repetition
No help available

set_start(*start_index: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:STARt
driver.source.gprf.generator.listPy.set_start(start_index = 1)
```

No command help available

param start_index
No help available

set_stop(*stop_index: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:STOP
driver.source.gprf.generator.listPy.set_stop(stop_index = 1)
```

No command help available

param stop_index
No help available

set_value(*enable_list_mode: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST
driver.source.gprf.generator.listPy.set_value(enable_list_mode = False)
```

No command help available

param enable_list_mode
No help available

Cloning the Group

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

Subgroups

6.13.1.1.4.1 Dgain

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:DGAin
SOURce:GPRF:GENerator<Instance>:LIST:DGAin:ALL
```

class DgainCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DGAin
value: float = driver.source.gprf.generator.listPy.dgain.get(index = 1)
```

No command help available

param index
No help available

return
digital_gain: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DGAin:ALL
value: List[float] = driver.source.gprf.generator.listPy.dgain.get_all()
```

No command help available

return
all_digital_gains: No help available

set(*index: int*, *digital_gain: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DGAin
driver.source.gprf.generator.listPy.dgain.set(index = 1, digital_gain = 1.0)
```

No command help available

param index
No help available

param digital_gain
No help available

set_all(*all_digital_gains: List[float]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DGAin:ALL
driver.source.gprf.generator.listPy.dgain.set_all(all_digital_gains = [1.1, 2.2,
↪ 3.3])
```

No command help available

param all_digital_gains
No help available

6.13.1.1.4.2 Dtime

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:DTIMe
SOURce:GPRF:GENerator<Instance>:LIST:DTIMe:ALL
```

class DtimeCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DTIMe
value: float = driver.source.gprf.generator.listPy.dtime.get(index = 1)
```

No command help available

param index
No help available

return
dwell_time: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DTIMe:ALL
value: List[float] = driver.source.gprf.generator.listPy.dtime.get_all()
```

No command help available

return
all_dwelltimes: No help available

set(*index: int*, *dwell_time: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DTIMe
driver.source.gprf.generator.listPy.dtime.set(index = 1, dwell_time = 1.0)
```

No command help available

param index
No help available

param dwell_time
No help available

set_all(*all_dwelltimes: List[float]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:DTIMe:ALL
driver.source.gprf.generator.listPy.dtime.set_all(all_dwelltimes = [1.1, 2.2, 3.
↪3])
```

No command help available

param all_dwelltimes
No help available

6.13.1.1.4.3 Esingle

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:LIST:ESINgle
```

class EsingleCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:ESINgle
driver.source.gprf.generator.listPy.esingle.set()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:ESINgle
driver.source.gprf.generator.listPy.esingle.set_with_opc()
```

No command help available

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

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

6.13.1.1.4.4 Fill

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:FILL:APPLy
SOURce:GPRF:GENerator<Instance>:LIST:FILL
```

class FillCls

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

class ValueStruct

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

- Start_Index: float: No parameter help available
- Range_Py: float: No parameter help available
- Index_Repetition: int: No parameter help available
- Start_Frequency: float: No parameter help available
- Freq_Increment: float: No parameter help available
- Start_Power: float: No parameter help available
- Power_Increment: float: No parameter help available
- Start_Dwell_Time: float: No parameter help available

- Reenable: bool: No parameter help available
- Modulation: bool: No parameter help available
- Start_Gain: float: No parameter help available
- Gain_Increment: float: No parameter help available

get_apply() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:FILL:APPLy
value: int = driver.source.gprf.generator.listPy.fill.get_apply()
```

No command help available

return
apply: No help available

set_value(*value: ValueStruct*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:FILL
structure = driver.source.gprf.generator.listPy.fill.ValueStruct()
structure.Start_Index: float = 1.0
structure.Range_Py: float = 1.0
structure.Index_Repetition: int = 1
structure.Start_Frequency: float = 1.0
structure.Freq_Increment: float = 1.0
structure.Start_Power: float = 1.0
structure.Power_Increment: float = 1.0
structure.Start_Dwell_Time: float = 1.0
structure.Reenable: bool = False
structure.Modulation: bool = False
structure.Start_Gain: float = 1.0
structure.Gain_Increment: float = 1.0
driver.source.gprf.generator.listPy.fill.set_value(value = structure)
```

No command help available

param value
see the help for ValueStruct structure arguments.

6.13.1.1.4.5 Frequency

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:FREQuency
SOURce:GPRF:GENerator<Instance>:LIST:FREQuency:ALL
```

class FrequencyCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:FREQuency
value: float = driver.source.gprf.generator.listPy.frequency.get(index = 1)
```

No command help available

param index
No help available

return
frequency: No help available

`get_all()` → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:FREQuency:ALL
value: List[float] = driver.source.gprf.generator.listPy.frequency.get_all()
```

No command help available

return
all_frequencies: No help available

`set(index: int, frequency: float)` → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:FREQuency
driver.source.gprf.generator.listPy.frequency.set(index = 1, frequency = 1.0)
```

No command help available

param index
No help available

param frequency
No help available

`set_all(all_frequencies: List[float])` → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:FREQuency:ALL
driver.source.gprf.generator.listPy.frequency.set_all(all_frequencies = [1.1, 2.
↪2, 3.3])
```

No command help available

param all_frequencies
No help available

6.13.1.1.4.6 Increment

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:INCRement:CATalog
SOURce:GPRF:GENerator<Instance>:LIST:INCRement
```

`class IncrementCls`

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

`get_catalog()` → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:INCRement:CATalog
value: List[str] = driver.source.gprf.generator.listPy.increment.get_catalog()
```

No command help available

return
list_incr_srcs: No help available

get_value() → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:INCRement
value: str = driver.source.gprf.generator.listPy.increment.get_value()
```

No command help available

return
list_incr_src: No help available

set_value(*list_incr_src: str*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:INCRement
driver.source.gprf.generator.listPy.increment.set_value(list_incr_src = 'abc')
```

No command help available

param list_incr_src
No help available

Cloning the Group

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

Subgroups

6.13.1.1.4.7 Enabling

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:INCRement:ENABling:CATalog
SOURce:GPRF:GENerator<Instance>:LIST:INCRement:ENABling
```

class EnablingCls

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

get_catalog() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:INCRement:ENABling:CATalog
value: List[str] = driver.source.gprf.generator.listPy.increment.enabling.get_
↪catalog()
```

No command help available

return
enabling_srcs: No help available

get_value() → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:INCRement:ENABling
value: str = driver.source.gprf.generator.listPy.increment.enabling.get_value()
```

No command help available

return
enabling: No help available

set_value(*enabling: str*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:INCRement:ENABling
driver.source.gprf.generator.listPy.increment.enabling.set_value(enabling = 'abc
↪')
```

No command help available

param enabling
No help available

6.13.1.1.4.8 Irepetition

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:IREPetition
SOURce:GPRF:GENerator<Instance>:LIST:IREPetition:ALL
```

class IrepetitionCls

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

get(*index: int*) → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:IREPetition
value: int = driver.source.gprf.generator.listPy.irepetition.get(index = 1)
```

No command help available

param index
No help available

return
repetition: No help available

get_all() → List[int]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:IREPetition:ALL
value: List[int] = driver.source.gprf.generator.listPy.irepetition.get_all()
```

No command help available

return
index_repetitions: No help available

set(*index: int*, *repetition: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:IREPetition
driver.source.gprf.generator.listPy.irepetition.set(index = 1, repetition = 1)
```

No command help available

param index
No help available

param repetition
No help available

set_all(*index_repetitions: List[int]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:IREPetition:ALL
driver.source.gprf.generator.listPy.irepetition.set_all(index_repetitions = [1,␣
↪2, 3])
```

No command help available

param index_repetitions
No help available

6.13.1.1.4.9 Modulation

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:MODulation
SOURce:GPRF:GENerator<Instance>:LIST:MODulation:ALL
```

class ModulationCls

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

get(*index: int*) → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:MODulation
value: bool = driver.source.gprf.generator.listPy.modulation.get(index = 1)
```

No command help available

param index
No help available

return
modulation: No help available

get_all() → List[bool]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:MODulation:ALL
value: List[bool] = driver.source.gprf.generator.listPy.modulation.get_all()
```

No command help available

return
all_modulations: No help available

set(*index: int*, *modulation: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:MODulation
driver.source.gprf.generator.listPy.modulation.set(index = 1, modulation =␣
↪False)
```

No command help available

param index
No help available

param modulation
No help available

set_all(*all_modulations: List[bool]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:MODulation:ALL
driver.source.gprf.generator.listPy.modulation.set_all(all_modulations = [True,␣
↪False, True])
```

No command help available

param all_modulations
No help available

6.13.1.1.4.10 Reenabling

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:REENabling
SOURce:GPRF:GENerator<Instance>:LIST:REENabling:ALL
```

class ReenablingCls

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

get(*index: int*) → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:REENabling
value: bool = driver.source.gprf.generator.listPy.reenabling.get(index = 1)
```

No command help available

param index
No help available

return
reenabling: No help available

get_all() → List[bool]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:REENabling:ALL
value: List[bool] = driver.source.gprf.generator.listPy.reenabling.get_all()
```

No command help available

return
all_reenables: No help available

set(*index: int*, *reenabling: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:REENabling
driver.source.gprf.generator.listPy.reenabling.set(index = 1, reenabling =␣
↪False)
```

No command help available

param index
No help available

param reenabling
No help available

set_all(*all_reenables: List[bool]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:REENabling:ALL
driver.source.gprf.generator.listPy.reenabling.set_all(all_reenables = [True,␣
↪False, True])
```

No command help available

param all_reenables
No help available

6.13.1.1.4.11 RfLevel

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:LIST:RFLevel
SOURce:GPRF:GENerator<Instance>:LIST:RFLevel:ALL
```

class RfLevelCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:RFLevel
value: float or bool = driver.source.gprf.generator.listPy.rfLevel.get(index =␣
↪1)
```

No command help available

param index
No help available

return
level: (float or boolean) No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:RFLevel:ALL
value: List[float or bool] = driver.source.gprf.generator.listPy.rfLevel.get_
↪all()
```

No command help available

return
all_levels: (float or boolean items) No help available

set(*index: int*, *level: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:RFLevel
driver.source.gprf.generator.listPy.rfLevel.set(index = 1, level = 1.0)
```

No command help available

param index
No help available

param level
(float or boolean) No help available

set_all(*all_levels: List[float]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:RFLevel:ALL
driver.source.gprf.generator.listPy.rfLevel.set_all(all_levels = [1.1, True, 2.
↪2, False, 3.3])
```

No command help available

param all_levels
(float or boolean items) No help available

6.13.1.1.4.12 Rlist

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:LIST:RLISt
```

class RlistCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:RLISt
driver.source.gprf.generator.listPy.rlist.set()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:RLISt
driver.source.gprf.generator.listPy.rlist.set_with_opc()
```

No command help available

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

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

6.13.1.1.4.13 Slist

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:LIST:SLISt
```

class SlistCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:SLISt
driver.source.gprf.generator.listPy.slist.set()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:SLISt
driver.source.gprf.generator.listPy.slist.set_with_opc()
```

No command help available

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

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

6.13.1.1.4.14 Sstop

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:LIST:SSTop
```

class SstopCls

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

class GetStruct

Response structure. Fields:

- Start_Index: int: No parameter help available
- Stop_Index: int: No parameter help available

get() → GetStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:SSTop
value: GetStruct = driver.source.gprf.generator.listPy.sstop.get()
```

No command help available

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

set(*start_index: int*, *stop_index: int*, *goto_index: int = None*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:LIST:SSTop
driver.source.gprf.generator.listPy.sstop.set(start_index = 1, stop_index = 1,␣
↪goto_index = 1)
```

No command help available

param start_index
No help available

param stop_index
No help available

param goto_index
No help available

6.13.1.1.5 Reliability

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:RELiability
SOURce:GPRF:GENerator<Instance>:RELiability:ALL
```

class ReliabilityCls

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

class AllStruct

Structure for reading output parameters. Fields:

- Reliability: int: Reliability indicator
- Reliability_Msg: str: Reason for the reliability value. Empty string '' for reliability = 0.
- Reliability_Add_Info: str: No parameter help available

get(*details: str = None*) → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RELiability
value: str = driver.source.gprf.generator.reliability.get(details = 'abc')
```

No command help available

Suppressed linked return values: reliability

param details
No help available

return
reliability_msg: No help available

get_all() → AllStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RELiability:ALL
value: AllStruct = driver.source.gprf.generator.reliability.get_all()
```

Reports if and why there are problems generating the configured signal. For possible values, see 'Reliability indicator'.

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

6.13.1.1.6 RfSettings

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:RFSettings:LOFRequency
SOURce:GPRF:GENerator<Instance>:RFSettings:LOLevel
SOURce:GPRF:GENerator<Instance>:RFSettings:DGAin
SOURce:GPRF:GENerator<Instance>:RFSettings:PEPower
SOURce:GPRF:GENerator<Instance>:RFSettings:EATTenuation
SOURce:GPRF:GENerator<Instance>:RFSettings:FREQuency
SOURce:GPRF:GENerator<Instance>:RFSettings:LEVel
```

class RfSettingsCls

RfSettings commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_dgain() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:DGAin
value: float = driver.source.gprf.generator.rfSettings.get_dgain()
```

Defines the digital gain of the RF generator.

return
digital_gain: No help available

get_eattenuation() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:EATTenuation
value: float = driver.source.gprf.generator.rfSettings.get_eattenuation()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the output connector.

return
ext_rf_out_att: No help available

get_frequency() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:FREQuency
value: float = driver.source.gprf.generator.rfSettings.get_frequency()
```

Sets the frequency of the unmodulated RF carrier. For the supported frequency range, see 'Frequency ranges'.

return
frequency: No help available

get_level() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:LEVel
value: float = driver.source.gprf.generator.rfSettings.get_level()
```

Sets the base RMS level of the RF generator.

return
level: No help available

get_lo_frequency() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:LOFRequency
value: float = driver.source.gprf.generator.rfSettings.get_lo_frequency()
```

Queries the required external LO frequency resulting from the generator settings. The command also triggers a refresh of the information before the query. So no need for a separate refresh command.

return
lo_frequency: No help available

get_lo_level() → LoLevel

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:LOLevel
value: enums.LoLevel = driver.source.gprf.generator.rfSettings.get_lo_level()
```

Queries whether the level of an external LO signal is correct.

return
lo_level: Level correct, too low, too high.

get_pe_power() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:PEPower
value: float = driver.source.gprf.generator.rfSettings.get_pe_power()
```

Queries the peak envelope power.

return
peak_envelope_pow: No help available

set_dgain(*digital_gain: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:DGAin
driver.source.gprf.generator.rfSettings.set_dgain(digital_gain = 1.0)
```

Defines the digital gain of the RF generator.

param digital_gain
No help available

set_eattenuation(*ext_rf_out_att: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:EATTenuation
driver.source.gprf.generator.rfSettings.set_eattenuation(ext_rf_out_att = 1.0)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the output connector.

param ext_rf_out_att
No help available

set_frequency(*frequency: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:FREQuency
driver.source.gprf.generator.rfSettings.set_frequency(frequency = 1.0)
```

Sets the frequency of the unmodulated RF carrier. For the supported frequency range, see 'Frequency ranges'.

param frequency
No help available

set_level(*level: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:RFSettings:LEVel
driver.source.gprf.generator.rfSettings.set_level(level = 1.0)
```

Sets the base RMS level of the RF generator.

param level
No help available

6.13.1.1.7 Sequencer

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:REPetition
SOURce:GPRF:GENerator<Instance>:SEQuencer:NREPetition
SOURce:GPRF:GENerator<Instance>:SEQuencer:RCOunt
SOURce:GPRF:GENerator<Instance>:SEQuencer:SIGNal
SOURce:GPRF:GENerator<Instance>:SEQuencer:CENTry
SOURce:GPRF:GENerator<Instance>:SEQuencer:UOPTions
```

class SequencerCls

Sequencer commands group definition. 105 total commands, 10 Subgroups, 6 group commands

get_centry() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:CENTry
value: int = driver.source.gprf.generator.sequencer.get_centry()
```

Queries the index of the processed entry. The remote query takes between 2 ms and 3 ms, which introduces an uncertainty to the results.

return
current_entry: If the sequencer is not running, NAV is returned.

get_nrepetition() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:NREPetition
value: int = driver.source.gprf.generator.sequencer.get_nrepetition()
```

Defines how often the sequencer list is processed in SINGle repetition mode.

return
num_of_rep: No help available

get_rcount() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:RCOunt
value: int = driver.source.gprf.generator.sequencer.get_rcount()
```

Queries the number of completed repetitions of the sequencer list, for repetition mode SINGle.

return
repcount: If the sequencer is not running or the repetition mode is CONTinuous, NAV is returned.

get_repetition() → RepeatMode

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:REPetition
value: enums.RepeatMode = driver.source.gprf.generator.sequencer.get_
↪repetition()
```

Defines the repetition mode for the sequencer list.

return
repetition: CONTinuous: unlimited repetitions, with cyclic processing SINGle: configured number of repetitions, with cyclic processing

get_signal() → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:SIGNal
value: bool = driver.source.gprf.generator.sequencer.get_signal()
```

Queries whether a signal is generated or not.

return
signal: No help available

get_uoptions() → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:UOPTions
value: str = driver.source.gprf.generator.sequencer.get_uoptions()
```

Queries a list of the used software options.

return
used_options: The string contains a comma-separated list of options. If the sequencer is OFF, NAV is returned. If the sequencer is not OFF but no options are used by the sequencer list, 'None' is returned.

set_nrepetition(*num_of_rep: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:NREPetition
driver.source.gprf.generator.sequencer.set_nrepetition(num_of_rep = 1)
```

Defines how often the sequencer list is processed in SINGle repetition mode.

param num_of_rep
No help available

set_repetition(*repetition: RepeatMode*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:REPetition
driver.source.gprf.generator.sequencer.set_repetition(repetition = enums.
↪RepeatMode.CONTinuous)
```

Defines the repetition mode for the sequencer list.

param repetition
CONTinuous: unlimited repetitions, with cyclic processing SINGle: configured number of repetitions, with cyclic processing

Cloning the Group

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

Subgroups

6.13.1.1.7.1 Apool

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:VALid
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:LOADed
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RREQuired
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RTOTal
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:FILE
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:REMove
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CLEar
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:MINDex
```

class ApoolCls

Apool commands group definition. 32 total commands, 13 Subgroups, 8 group commands

clear() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CLEar
driver.source.gprf.generator.sequencer.apool.clear()
```

Removes all files from the ARB file pool.

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CLEar
driver.source.gprf.generator.sequencer.apool.clear_with_opc()
```

Removes all files from the ARB file pool.

Same as clear, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.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_loaded() → YesNoStatus

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:LOADed
value: enums.YesNoStatus = driver.source.gprf.generator.sequencer.apool.get_
↪loaded()
```

Queries whether the ARB file pool is downloaded to the ARB RAM.

return
loaded: No help available

get_mindex() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:MINDex
value: int = driver.source.gprf.generator.sequencer.apool.get_mindex()
```

Queries the highest index of the ARB file pool. The pool contains files with the indices 0 to <MaximumIndex>.

return
maximum_index: Highest index. If the file pool is empty, NAV is returned.

get_rrequired() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RREQuired
value: float = driver.source.gprf.generator.sequencer.apool.get_rrequired()
```

Queries the amount of RAM required by the ARB files in the pool.

return
ram_required: No help available

get_rtotal() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RTOTal
value: float = driver.source.gprf.generator.sequencer.apool.get_rtotal()
```

Queries the amount of RAM available for ARB files.

return
ram_total: No help available

get_valid() → YesNoStatus

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:VALid
value: enums.YesNoStatus = driver.source.gprf.generator.sequencer.apool.get_
↪valid()
```

Queries whether the ARB file pool is valid.

return
valid: No help available

set_file(*arb_file: str*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:FILE
driver.source.gprf.generator.sequencer.apool.set_file(arb_file = 'abc')
```

Adds an ARB file to the ARB file pool.

param arb_file
Path and filename Example: '@WAVEFORM/myARBfile.wv'

set_remove(*indices: List[int]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:REMove
driver.source.gprf.generator.sequencer.apool.set_remove(indices = [1, 2, 3])
```

Removes selected files from the ARB file pool.

param indices
Indices of the files to be removed. You can specify a single index or a comma-separated list of indices.

Cloning the Group

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

Subgroups

6.13.1.1.7.2 Check

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CHECk
```

class CheckCls

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

get(*index: int*) → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CHECk
value: bool = driver.source.gprf.generator.sequencer.apool.check.get(index = 1)
```

No command help available

param index
No help available

return
check: No help available

set(*index: int*, *check: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CHECk
driver.source.gprf.generator.sequencer.apool.check.set(index = 1, check = False)
```

No command help available

param index
No help available

param check
No help available

6.13.1.1.7.3 CrcProtect

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CRCProtect:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CRCProtect
```

class CrcProtectCls

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

get(*index: int*) → YesNoStatus

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CRCProtect
value: enums.YesNoStatus = driver.source.gprf.generator.sequencer.apool.
↪crcProtect.get(index = 1)
```

Queries whether the ARB file with the specified <Index> contains a CRC checksum.

param index
No help available

return
crc_protection: No help available

get_all() → List[YesNoStatus]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:CRCProtect:ALL
value: List[enums.YesNoStatus] = driver.source.gprf.generator.sequencer.apool.
↪crcProtect.get_all()
```

Queries whether the ARB files in the file pool contain CRC checksums.

return
crc_protection: Comma-separated list of values, one value per file

6.13.1.1.7.4 Download

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:DOWNload
```

class DownloadCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:DOWNload
driver.source.gprf.generator.sequencer.apool.download.set()
```

Downloads the ARB files from the file pool to the ARB RAM.

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:DOWNload
driver.source.gprf.generator.sequencer.apool.download.set_with_opc()
```

Downloads the ARB files from the file pool to the ARB RAM.

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

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

6.13.1.1.7.5 Duration

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:DURation:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:DURation
```

class DurationCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:DURation
value: float = driver.source.gprf.generator.sequencer.apool.duration.get(index␣
↪= 1)
```

Queries the duration of the ARB file with the specified <Index>.

param index
No help available

return
duration: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:DURation:ALL
value: List[float] = driver.source.gprf.generator.sequencer.apool.duration.get_
↪all()
```

Queries the durations of the ARB files in the file pool.

return
duration: Comma-separated list of values, one value per file

6.13.1.1.7.6 Paratio

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PARatio:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PARatio
```

class ParatioCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PARatio
value: float = driver.source.gprf.generator.sequencer.apool.paratio.get(index =␣
↪1)
```

Queries the peak to average ratio of the ARB file with the specified <Index>.

param index
No help available

return
peak_avg_ratio: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PARatio:ALL
value: List[float] = driver.source.gprf.generator.sequencer.apool.paratio.get_
↪all()
```

Queries the peak to average ratios of the ARB files in the file pool.

return
peak_avg_ratio: Comma-separated list of values, one value per file

6.13.1.1.7.7 Path

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PATH:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PATH
```

class PathCls

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

get(*index: int*) → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PATH
value: str = driver.source.gprf.generator.sequencer.apool.path.get(index = 1)
```

Queries the path and filename of the ARB file with the specified <Index>.

param index
No help available

return
full_path_name: Absolute path and name of the file.

get_all() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:PATH:ALL
value: List[str] = driver.source.gprf.generator.sequencer.apool.path.get_all()
```

Queries the path and filename of the ARB files in the file pool.

return
path: Comma-separated list of strings, one string per file Each string contains the complete path and name of a file.

6.13.1.1.7.8 Poffset

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:POFFset:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:POFFset
```

class PoffsetCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:POFFset
value: float = driver.source.gprf.generator.sequencer.apool.poffset.get(index =␣
↪1)
```

Queries the peak offset of the ARB file with the specified <Index>.

param index
No help available

return
peak_offset: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:POFFset:ALL
value: List[float] = driver.source.gprf.generator.sequencer.apool.poffset.get_
↪all()
```

Queries the peak offsets of the ARB files in the file pool.

return
peak_offset: Comma-separated list of values, one value per file

6.13.1.1.7.9 Reliability

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RELiability:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RELiability
```

class ReliabilityCls

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

get(*index: int*) → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RELiability
value: int = driver.source.gprf.generator.sequencer.apool.reliability.get(index␣
↪= 1)
```

Queries the reliability indicator for the ARB file with the specified <Index>. For possible values, see 'Reliability indicator'.

param index
No help available

return
reliability: Reliability indicator

`get_all()` → List[int]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RELiability:ALL
value: List[int] = driver.source.gprf.generator.sequencer.apool.reliability.get_
↪all()
```

Queries the reliability indicators for all ARB files in the file pool. For possible values, see 'Reliability indicator'.

return
reliability: Comma-separated list of values One value per file, from index 0 to index n

6.13.1.1.7.10 Rmessage

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RMESsage:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RMESsage
```

`class RmessageCls`

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

`get(index: int)` → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RMESsage
value: str = driver.source.gprf.generator.sequencer.apool.rmessage.get(index =␣
↪1)
```

Queries the reliability message for the ARB file with the specified <Index>. For possible values, see 'Reliability indicator'.

param index
No help available

return
reliability_msg: Reliability message

`get_all()` → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:RMESsage:ALL
value: List[str] = driver.source.gprf.generator.sequencer.apool.rmessage.get_
↪all()
```

Queries the reliability messages for all ARB files in the file pool. For possible values, see 'Reliability indicator'.

return
reliability_msg: Comma-separated list of strings One string per file, from index 0 to index n

6.13.1.1.7.11 Roption

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:ROPTion:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:ROPTion
```

class RoptionCls

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

get(*index: int*) → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:ROPTion
value: str = driver.source.gprf.generator.sequencer.apool.roption.get(index = 1)
```

Queries the options required by the ARB file with the specified <Index>.

param index
No help available

return
required_options: No help available

get_all() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:ROPTion:ALL
value: List[str] = driver.source.gprf.generator.sequencer.apool.roption.get_
↪all()
```

Queries the options required by the ARB files in the file pool.

return
required_options: Comma-separated list of strings, one string per file

6.13.1.1.7.12 Samples

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SAMPles:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SAMPles
```

class SamplesCls

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

get(*index: int*) → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SAMPles
value: int = driver.source.gprf.generator.sequencer.apool.samples.get(index = 1)
```

Queries the number of samples in the ARB file with the specified <Index>.

param index
No help available

return
samples: Number of samples

`get_all()` → List[int]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SAMPles:ALL
value: List[int] = driver.source.gprf.generator.sequencer.apool.samples.get_
↪all()
```

Queries the numbers of samples in the ARB files of the file pool.

return
samples: Comma-separated list of values, one value per file

6.13.1.1.7.13 SymbolRate

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SRATe:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SRATe
```

`class SymbolRateCls`

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

`get`(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SRATe
value: float = driver.source.gprf.generator.sequencer.apool.symbolRate.
↪get(index = 1)
```

Queries the sample rate of the ARB file with the specified <Index>.

param index
No help available

return
sample_rate: No help available

`get_all()` → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:SRATe:ALL
value: List[float] = driver.source.gprf.generator.sequencer.apool.symbolRate.
↪get_all()
```

Queries the sample rates of the ARB files in the file pool.

return
sample_rate: Comma-separated list of values, one value per file

6.13.1.1.7.14 Waveform

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:WAVeform:ALL
SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:WAVeform
```

class WaveformCls

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

get(*index: int*) → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:WAVeform
value: str = driver.source.gprf.generator.sequencer.apool.waveform.get(index =␣
↪1)
```

Queries the name of the ARB file with the specified <Index>.

param index
No help available

return
waveform: Filename (without path) .

get_all() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:APOol:WAVeform:ALL
value: List[str] = driver.source.gprf.generator.sequencer.apool.waveform.get_
↪all()
```

Queries the names of the ARB files in the file pool.

return
waveform: Comma-separated list of strings One string per file, from index 0 to index n

6.13.1.1.7.15 Dtone

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:DTONe:RATio
```

class DtoneCls

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

get_ratio() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:DTONe:RATio
value: float = driver.source.gprf.generator.sequencer.dtone.get_ratio()
```

Specifies the ratio in dB between the RMS levels of the two signals.

return
ratio: No help available

set_ratio(*ratio: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:DTONe:RATio
driver.source.gprf.generator.sequencer.dtone.set_ratio(ratio = 1.0)
```

Specifies the ratio in dB between the RMS levels of the two signals.

param ratio
No help available

Cloning the Group

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

Subgroups

6.13.1.1.7.16 Ofrequency<FrequencySource>

RepCap Settings

```
# Range: Src1 .. Src2
rc = driver.source.gprf.generator.sequencer.dtone.ofrequency.repcap_frequencySource_get()
driver.source.gprf.generator.sequencer.dtone.ofrequency.repcap_frequencySource_
↪set(repcap.FrequencySource.Src1)
```

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:DTONe:OFRequency<source>
```

class OfrequencyCls

Ofrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: FrequencySource, default value after init: FrequencySource.Src1

get(*frequencySource=FrequencySource.Default*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:DTONe:OFRequency<source>
value: float = driver.source.gprf.generator.sequencer.dtone.ofrequency.
↪get(frequencySource = repcap.FrequencySource.Default)
```

Selects an offset frequency. The frequency of the modulated signal is equal to the generator frequency plus the offset.

param frequencySource
optional repeated capability selector. Default value: Src1 (settable in the interface 'Ofrequency')

return
frequency: No help available

set(*frequency: float*, *frequencySource=FrequencySource.Default*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:DTONe:OFRequency<source>
driver.source.gprf.generator.sequencer.dtone.ofrequency.set(frequency = 1.0,␣
↪frequencySource = repcap.FrequencySource.Default)
```

Selects an offset frequency. The frequency of the modulated signal is equal to the generator frequency plus the offset.

param frequency
No help available

param frequencySource
optional repeated capability selector. Default value: Src1 (settable in the interface 'Ofrequency')

Cloning the Group

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

6.13.1.1.7.17 ListPy

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:CREate
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:INDex
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:MINDex
```

class ListPyCls

ListPy commands group definition. 54 total commands, 13 Subgroups, 3 group commands

get_index() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:INDex
value: int = driver.source.gprf.generator.sequencer.listPy.get_index()
```

Selects an entry of the sequencer list. Some other commands use this setting.

return
current_index: Index of the selected list entry

get_mindex() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:MINDex
value: int = driver.source.gprf.generator.sequencer.listPy.get_mindex()
```

Queries the highest index of the sequencer list. The list contains entries with the indices 0 to <MaximumIndex>.

return
maximum_index: No help available

set_create(*entries: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:CREate
driver.source.gprf.generator.sequencer.listPy.set_create(entries = 1.0)
```

Deletes all entries of the sequencer list and creates the defined number of new entries with default settings.

param entries
Number of entries to be created.

set_index(*current_index: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:INDex
driver.source.gprf.generator.sequencer.listPy.set_index(current_index = 1)
```

Selects an entry of the sequencer list. Some other commands use this setting.

param current_index
Index of the selected list entry

Cloning the Group

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

Subgroups

6.13.1.1.7.18 Acycles

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ACYCles
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ACYCles:ALL
```

class AcyclesCls

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

get(*index: int*) → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ACYCles
value: int = driver.source.gprf.generator.sequencer.listPy.acycles.get(index =␣
↪1)
```

Defines or queries the number of ARB cycles for the sequencer list entry with the selected <Index>.

param index
No help available

return
arb_cycles: No help available

get_all() → List[int]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ACYCles:ALL
value: List[int] = driver.source.gprf.generator.sequencer.listPy.acycles.get_
↪all()
```

Defines the ARB cycles for all sequencer list entries.

return
arb_cycles: Comma-separated list of values, one value per list entry

set(*index: int*, *arb_cycles: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ACYCles
driver.source.gprf.generator.sequencer.listPy.acycles.set(index = 1, arb_cycles␣
↪= 1)
```

Defines or queries the number of ARB cycles for the sequencer list entry with the selected <Index>.

param index
No help available

param arb_cycles
No help available

set_all(*arb_cycles: List[int]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ACYCles:ALL
driver.source.gprf.generator.sequencer.listPy.acycles.set_all(arb_cycles = [1,␣
↪2, 3])
```

Defines the ARB cycles for all sequencer list entries.

param arb_cycles
Comma-separated list of values, one value per list entry

6.13.1.1.7.19 Dgain

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DGAin
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DGAin:ALL
```

class DgainCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DGAin
value: float = driver.source.gprf.generator.sequencer.listPy.dgain.get(index =␣
↪1)
```

Defines or queries the digital gain for the sequencer list entry with the selected <Index>.

param index
No help available

return
digital_gain: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DGAin:ALL
value: List[float] = driver.source.gprf.generator.sequencer.listPy.dgain.get_
↪all()
```

Defines the digital gains for all sequencer list entries.

return
digital_gain: Comma-separated list of values, one value per list entry

set(*index: int*, *digital_gain: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DGAin
driver.source.gprf.generator.sequencer.listPy.dgain.set(index = 1, digital_gain␣
↪= 1.0)
```

Defines or queries the digital gain for the sequencer list entry with the selected <Index>.

param index
No help available

param digital_gain
No help available

set_all(*digital_gain: List[float]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DGAin:ALL
driver.source.gprf.generator.sequencer.listPy.dgain.set_all(digital_gain = [1.1,
↪ 2.2, 3.3])
```

Defines the digital gains for all sequencer list entries.

param digital_gain
Comma-separated list of values, one value per list entry

6.13.1.1.7.20 Dtime

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DTIMe
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DTIMe:ALL
```

class DtimeCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DTIMe
value: float = driver.source.gprf.generator.sequencer.listPy.dtime.get(index =␣
↪1)
```

Defines or queries the duration for the sequencer list entry with the selected <Index>.

param index
No help available

return
dwell_time: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DTIMe:ALL
value: List[float] = driver.source.gprf.generator.sequencer.listPy.dtime.get_
↪all()
```

Defines the duration for all sequencer list entries.

return
dwell_time: Comma-separated list of values, one value per list entry

set(*index: int*, *dwell_time: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DTIMe
driver.source.gprf.generator.sequencer.listPy.dtime.set(index = 1, dwell_time =␣
↪1.0)
```

Defines or queries the duration for the sequencer list entry with the selected <Index>.

param index
No help available

param dwell_time
No help available

set_all(*dwell_time: List[float]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:DTIMe:ALL
driver.source.gprf.generator.sequencer.listPy.dtime.set_all(dwell_time = [1.1,␣
↪2.2, 3.3])
```

Defines the duration for all sequencer list entries.

param dwell_time
Comma-separated list of values, one value per list entry

6.13.1.1.7.21 Entry

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:DELete
```

class EntryCls

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

delete(*index: int = None*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:DELete
driver.source.gprf.generator.sequencer.listPy.entry.delete(index = 1)
```

Deletes the selected entry from the sequencer list. You can specify <Index> to select that entry. Or you can select an entry via method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.index. After the deletion, the selection moves to the next entry, if possible. Otherwise, it moves to the previous entry.

param index
Index of the entry to be deleted.

Cloning the Group

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

Subgroups

6.13.1.1.7.22 Call

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:CALL
```

class CallCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:CALL
driver.source.gprf.generator.sequencer.listPy.entry.call.set()
```

Deletes all entries of the sequencer list and creates an entry with default settings.

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:CALL
driver.source.gprf.generator.sequencer.listPy.entry.call.set_with_opc()
```

Deletes all entries of the sequencer list and creates an entry with default settings.

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

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

6.13.1.1.7.23 Insert

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:INSert
```

class InsertCls

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

set(*after_index: int = None*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:INSert
driver.source.gprf.generator.sequencer.listPy.entry.insert.set(after_index = 1)
```

Inserts a new entry after the selected entry into the sequencer list. You can specify <AfterIndex> to select that entry. Or you can select an entry via method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.index.

param after_index
Index of the entry to be selected.

6.13.1.1.7.24 Mdown

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:MDOWn
```

class MdownCls

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

set(*index: int = None*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:MDOWn
driver.source.gprf.generator.sequencer.listPy.entry.mdown.set(index = 1)
```

Moves the selected entry of the sequencer list one position down. You can specify <Index> to select that entry. Or you can select an entry via method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.index. The selection moves with the entry.

param index
Index of the entry to be moved.

6.13.1.1.7.25 Mup

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:MUP
```

class MupCls

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

set(*index: int = None*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ENTRy:MUP
driver.source.gprf.generator.sequencer.listPy.entry.mup.set(index = 1)
```

Moves the selected entry of the sequencer list one position up. You can specify <Index> to select that entry. Or you can select an entry via method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.index. The selection moves with the entry.

param index
Index of the entry to be moved.

6.13.1.1.7.26 Fill

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:SINDex
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:RANGe
```

class FillCls

Fill commands group definition. 12 total commands, 4 Subgroups, 2 group commands

get_range() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:RANGe
value: int = driver.source.gprf.generator.sequencer.listPy.fill.get_range()
```

Specifies the number of entries to be filled.

return
range_py: No help available

get_sindex() → int

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:SINDex
value: int = driver.source.gprf.generator.sequencer.listPy.fill.get_sindex()
```

Selects the first index of the sequence to be filled.

return
start_index: No help available

set_range(*range_py: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:RANGe
driver.source.gprf.generator.sequencer.listPy.fill.set_range(range_py = 1)
```

Specifies the number of entries to be filled.

param range_py
No help available

set_sindex(*start_index: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:SINDex
driver.source.gprf.generator.sequencer.listPy.fill.set_sindex(start_index = 1)
```

Selects the first index of the sequence to be filled.

param start_index
No help available

Cloning the Group

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

Subgroups

6.13.1.1.7.27 Apply

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:APPLy
```

class ApplyCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:APPLy
driver.source.gprf.generator.sequencer.listPy.fill.apply.set()
```

Fills the sequencer list with a sequence of entries, as configured by the other SOURce:GPRF:GEN<i>:SEQuencer:LIST:FILL:... commands.

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:APPLy
driver.source.gprf.generator.sequencer.listPy.fill.apply.set_with_opc()
```

Fills the sequencer list with a sequence of entries, as configured by the other SOURce:GPRF:GEN<i>:SEQuencer:LIST:FILL:... commands.

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

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

6.13.1.1.7.28 Dgain

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:SVALue
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:INCRement
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:KEEP
```

class DgainCls

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

get_increment() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:INCRement
value: float = driver.source.gprf.generator.sequencer.listPy.fill.dgain.get_
↪increment()
```

Configures the increment for filling the sequencer list with digital gain values.

return
increment: No help available

get_keep() → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:KEEP
value: bool = driver.source.gprf.generator.sequencer.listPy.fill.dgain.get_
↪keep()
```

Selects whether the digital gain of existing entries is kept or overwritten when the sequencer list is filled.

return
keep_flag: OFF: overwrite values ON: keep values

get_svalue() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:SVALue
value: float = driver.source.gprf.generator.sequencer.listPy.fill.dgain.get_
↪svalue()
```

Configures the start value for filling the sequencer list with digital gain values.

return
start_value: No help available

set_increment(*increment: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:INCRement
driver.source.gprf.generator.sequencer.listPy.fill.dgain.set_
↪increment(increment = 1.0)
```

Configures the increment for filling the sequencer list with digital gain values.

param increment
No help available

set_keep(*keep_flag: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:KEEP
driver.source.gprf.generator.sequencer.listPy.fill.dgain.set_keep(keep_flag =␣
↪False)
```

Selects whether the digital gain of existing entries is kept or overwritten when the sequencer list is filled.

param keep_flag
OFF: overwrite values ON: keep values

set_svalue(*start_value: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:DGAin:SVALue
driver.source.gprf.generator.sequencer.listPy.fill.dgain.set_svalue(start_value␣
↪= 1.0)
```

Configures the start value for filling the sequencer list with digital gain values.

param start_value
No help available

6.13.1.1.7.29 Frequency

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:SVALue
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:INCRement
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:KEEP
```

class FrequencyCls

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

get_increment() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:INCRement
value: float = driver.source.gprf.generator.sequencer.listPy.fill.frequency.get_
↪increment()
```

Configures the increment for filling the sequencer list with frequency values.

return
increment: No help available

get_keep() → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:KEEP
value: bool = driver.source.gprf.generator.sequencer.listPy.fill.frequency.get_
↪keep()
```

Selects whether the frequency of existing entries is kept or overwritten when the sequencer list is filled.

return
keep_flag: OFF: overwrite values ON: keep values

get_svalue() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:SVALue
value: float = driver.source.gprf.generator.sequencer.listPy.fill.frequency.get_
↪svalue()
```

Configures the start value for filling the sequencer list with frequency values. For the supported frequency range, see 'Frequency ranges'.

return
start_value: No help available

set_increment(*increment: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:INCRement
driver.source.gprf.generator.sequencer.listPy.fill.frequency.set_
↪increment(increment = 1.0)
```

Configures the increment for filling the sequencer list with frequency values.

param increment
No help available

set_keep(*keep_flag: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:KEEP
driver.source.gprf.generator.sequencer.listPy.fill.frequency.set_keep(keep_flag=False)
```

Selects whether the frequency of existing entries is kept or overwritten when the sequencer list is filled.

param keep_flag
OFF: overwrite values ON: keep values

set_svalue(*start_value: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:FREQuency:SVALue
driver.source.gprf.generator.sequencer.listPy.fill.frequency.set_svalue(start_value = 1.0)
```

Configures the start value for filling the sequencer list with frequency values. For the supported frequency range, see 'Frequency ranges'.

param start_value
No help available

6.13.1.1.7.30 Lrms

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:SVALue
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:INCRement
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:KEEP
```

class LrmsCls

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

get_increment() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:INCRement
value: float = driver.source.gprf.generator.sequencer.listPy.fill.lrms.get_increment()
```

Configures the increment for filling the sequencer list with level values.

return
increment: No help available

get_keep() → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:KEEP
value: bool = driver.source.gprf.generator.sequencer.listPy.fill.lrms.get_keep()
```

Selects whether the level of existing entries is kept or overwritten when the sequencer list is filled.

return
keep_flag: OFF: overwrite values ON: keep values

get_svalue() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:SVALue
value: float = driver.source.gprf.generator.sequencer.listPy.fill.lrms.get_
↪svalue()
```

Configures the start value for filling the sequencer list with level values.

return
start_value: No help available

set_increment(*increment: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:INCRement
driver.source.gprf.generator.sequencer.listPy.fill.lrms.set_increment(increment␣
↪= 1.0)
```

Configures the increment for filling the sequencer list with level values.

param increment
No help available

set_keep(*keep_flag: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:KEEP
driver.source.gprf.generator.sequencer.listPy.fill.lrms.set_keep(keep_flag =␣
↪False)
```

Selects whether the level of existing entries is kept or overwritten when the sequencer list is filled.

param keep_flag
OFF: overwrite values ON: keep values

set_svalue(*start_value: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FILL:LRMS:SVALue
driver.source.gprf.generator.sequencer.listPy.fill.lrms.set_svalue(start_value␣
↪= 1.0)
```

Configures the start value for filling the sequencer list with level values.

param start_value
No help available

6.13.1.1.7.31 Frequency

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FREQuency
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FREQuency:ALL
```

class FrequencyCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FREQuency
value: float = driver.source.gprf.generator.sequencer.listPy.frequency.
↪get(index = 1)
```

Defines or queries the RF generator frequency for the sequencer list entry with the selected <Index>. For the supported frequency range, see 'Frequency ranges'.

param index
No help available

return
frequency: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FREQuency:ALL
value: List[float] = driver.source.gprf.generator.sequencer.listPy.frequency.
↪get_all()
```

Defines the RF generator frequencies for all sequencer list entries. For the supported frequency range, see 'Frequency ranges'.

return
frequency: Comma-separated list of values, one value per list entry

set(*index: int*, *frequency: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FREQuency
driver.source.gprf.generator.sequencer.listPy.frequency.set(index = 1,␣
↪frequency = 1.0)
```

Defines or queries the RF generator frequency for the sequencer list entry with the selected <Index>. For the supported frequency range, see 'Frequency ranges'.

param index
No help available

param frequency
No help available

set_all(*frequency: List[float]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:FREQuency:ALL
driver.source.gprf.generator.sequencer.listPy.frequency.set_all(frequency = [1.
↪1, 2.2, 3.3])
```

Defines the RF generator frequencies for all sequencer list entries. For the supported frequency range, see 'Frequency ranges'.

param frequency
Comma-separated list of values, one value per list entry

6.13.1.1.7.32 Itransition

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ITRansition
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ITRansition:ALL
```

class ItransitionCls

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

get(*index: int*) → IncTransition

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ITRansition
value: enums.IncTransition = driver.source.gprf.generator.sequencer.listPy.
↪itransition.get(index = 1)
```

Defines or queries a condition for the transition to the next list entry, for the sequencer list entry with the selected <Index>.

param index
No help available

return
inc_transition: IMMediate: immediately RMARker: restart marker WMA1 to WMA4: waveform markers 1 to 4

get_all() → List[IncTransition]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ITRansition:ALL
value: List[enums.IncTransition] = driver.source.gprf.generator.sequencer.
↪listPy.itransition.get_all()
```

Defines or queries a condition for the transition to the next list entry, for all sequencer list entries.

return
inc_transition: Comma-separated list of values, one value per list entry. Immediate, restart marker, waveform marker 1 to 4.

set(*index: int*, *inc_transition: IncTransition*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ITRansition
driver.source.gprf.generator.sequencer.listPy.itransition.set(index = 1, inc_
↪transition = enums.IncTransition.IMMediate)
```

Defines or queries a condition for the transition to the next list entry, for the sequencer list entry with the selected <Index>.

param index
No help available

param inc_transition
IMMediate: immediately RMARker: restart marker WMA1 to WMA4: waveform markers 1 to 4

set_all(*inc_transition: List[IncTransition]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:ITRansition:ALL
driver.source.gprf.generator.sequencer.listPy.itransition.set_all(inc_
↪transition = [IncTransition.IMMediate, IncTransition.WMA4])
```

Defines or queries a condition for the transition to the next list entry, for all sequencer list entries.

param inc_transition
Comma-separated list of values, one value per list entry. Immediate, restart marker, waveform marker 1 to 4.

6.13.1.1.7.33 Lincrement

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LINCrement
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LINCrement:ALL
```

class LincrementCls

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

get(*index: int*) → ListIncrement

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LINCrement
value: enums.ListIncrement = driver.source.gprf.generator.sequencer.listPy.
↪lincrement.get(index = 1)
```

Defines or queries the list increment for the sequencer list entry with the selected <Index>.

param index
No help available

return
list_increment: - DTIMe: The dwell time is defined via SOURce:GPRF:GENi:SEQuencer:LIST:DTIMe. - ACYCles: The ARB cycles are defined via SOURce:GPRF:GENi:SEQuencer:LIST:ACYCles. - USER: A user action is triggered via TRIGger:GPRF:GENi:SEQuencer:MANual:EXECute. - MEASurement: The measurement source is selected via TRIGger:GPRF:GENi:SEQuencer:ISMeas:SOURce. - TRIGger: The trigger source is selected via TRIGger:GPRF:GENi:SEQuencer:ISTRigger:SOURce.

get_all() → List[ListIncrement]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LINCrement:ALL
value: List[enums.ListIncrement] = driver.source.gprf.generator.sequencer.
↪listPy.lincrement.get_all()
```

Defines the list increments for all sequencer list entries.

return
list_increment: Comma-separated list of values, one value per list entry - DTIMe: The dwell time is defined via SOURce:GPRF:GENi:SEQuencer:LIST:DTIMe:ALL. - ACYCles: The ARB cycles are defined via SOURce:GPRF:GENi:SEQuencer:LIST:ACYCles:ALL. - USER: A user action is triggered via TRIGger:GPRF:GENi:SEQuencer:MANual:EXECute.

- MEASurement: The measurement source is selected via TRIGger:GPRF:GENi:SEQuencer:ISMeas:SOURce. - TRIGger: The trigger source is selected via TRIGger:GPRF:GENi:SEQuencer:ISTRigger:SOURce.

set(*index: int*, *list_increment: ListIncrement*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LINCrement
driver.source.gprf.generator.sequencer.listPy.lincrement.set(index = 1, list_
↪increment = enums.ListIncrement.ACYCles)
```

Defines or queries the list increment for the sequencer list entry with the selected <Index>.

param index
No help available

param list_increment

- DTIMe: The dwell time is defined via SOURce:GPRF:GENi:SEQuencer:LIST:DTIMe.
- ACYCles: The ARB cycles are defined via SOURce:GPRF:GENi:SEQuencer:LIST:ACYCles.
- USER: A user action is triggered via TRIGger:GPRF:GENi:SEQuencer:MANual:EXECute.
- MEASurement: The measurement source is selected via TRIGger:GPRF:GENi:SEQuencer:ISMeas:SOURce.
- TRIGger: The trigger source is selected via TRIGger:GPRF:GENi:SEQuencer:ISTRigger:SOURce.

set_all(*list_increment: List[ListIncrement]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LINCrement:ALL
driver.source.gprf.generator.sequencer.listPy.lincrement.set_all(list_increment␣
↪= [ListIncrement.ACYCles, ListIncrement.USER])
```

Defines the list increments for all sequencer list entries.

param list_increment
Comma-separated list of values, one value per list entry - DTIMe: The dwell time is defined via SOURce:GPRF:GENi:SEQuencer:LIST:DTIMe:ALL. - ACYCles: The ARB cycles are defined via SOURce:GPRF:GENi:SEQuencer:LIST:ACYCles:ALL. - USER: A user action is triggered via TRIGger:GPRF:GENi:SEQuencer:MANual:EXECute. - MEASurement: The measurement source is selected via TRIGger:GPRF:GENi:SEQuencer:ISMeas:SOURce. - TRIGger: The trigger source is selected via TRIGger:GPRF:GENi:SEQuencer:ISTRigger:SOURce.

6.13.1.1.7.34 Lrms

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LRMS
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LRMS:ALL
```

class LrmsCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LRMS
value: float = driver.source.gprf.generator.sequencer.listPy.lrms.get(index = 1)
```

Defines or queries the level for the sequencer list entry with the selected <Index>.

param index
No help available

return
level_rms: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LRMS:ALL
value: List[float] = driver.source.gprf.generator.sequencer.listPy.lrms.get_
↪all()
```

Defines the level for all sequencer list entries.

return
level_rms: Comma-separated list of values, one value per list entry

set(*index: int*, *level_rms: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LRMS
driver.source.gprf.generator.sequencer.listPy.lrms.set(index = 1, level_rms = 1.
↪0)
```

Defines or queries the level for the sequencer list entry with the selected <Index>.

param index
No help available

param level_rms
No help available

set_all(*level_rms: List[float]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:LRMS:ALL
driver.source.gprf.generator.sequencer.listPy.lrms.set_all(level_rms = [1.1, 2.
↪2, 3.3])
```

Defines the level for all sequencer list entries.

param level_rms
Comma-separated list of values, one value per list entry

6.13.1.1.7.35 Signal

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal
```

class SignalCls

Signal commands group definition. 13 total commands, 4 Subgroups, 1 group commands

get(*index: int*) → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal
value: str = driver.source.gprf.generator.sequencer.listPy.signal.get(index = 1)
```

Defines or queries the signal type for the sequencer list entry with the selected <Index>. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.Signal.Catalog.value.

param index
No help available

return
signal: Signal type

set(*index: int*, *signal: str*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal
driver.source.gprf.generator.sequencer.listPy.signal.set(index = 1, signal =
↪'abc')
```

Defines or queries the signal type for the sequencer list entry with the selected <Index>. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.Signal.Catalog.value.

param index
No help available

param signal
Signal type

Cloning the Group

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

Subgroups

6.13.1.1.7.36 All

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:OLD
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:CONTinue
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:WAVeform
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL
```

class AllCls

All commands group definition. 7 total commands, 3 Subgroups, 4 group commands

continue_py() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:CONTinue
driver.source.gprf.generator.sequencer.listPy.signal.all.continue_py()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:CONTinue
driver.source.gprf.generator.sequencer.listPy.signal.all.continue_py_with_opc()
```

No command help available

Same as continue_py, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.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_old() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:OLD
value: List[str] = driver.source.gprf.generator.sequencer.listPy.signal.all.get_
↪old()
```

No command help available

return
signal: No help available

get_value() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL
value: List[str] = driver.source.gprf.generator.sequencer.listPy.signal.all.get_
↪value()
```

Defines the signal types for all sequencer list entries. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.Signal.Catalog.value.

return
signal: Comma-separated list of strings, one string per list entry

set_old(*signal: List[str]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:OLD
driver.source.gprf.generator.sequencer.listPy.signal.all.set_old(signal = ['abc1
↪', 'abc2', 'abc3'])
```

No command help available

param signal
No help available

set_value(*signal: List[str]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL
driver.source.gprf.generator.sequencer.listPy.signal.all.set_value(signal = [
↪'abc1', 'abc2', 'abc3'])
```

Defines the signal types for all sequencer list entries. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Source.Gprf.Generator.Sequencer.ListPy.Signal.Catalog.value.

param signal
Comma-separated list of strings, one string per list entry

set_waveform(*waveform: str*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:WAVeform
driver.source.gprf.generator.sequencer.listPy.signal.all.set_waveform(waveform␣
↪= 'abc')
```

No command help available

param waveform
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.sequencer.listPy.signal.all.clone()
```

Subgroups

6.13.1.1.7.37 Cw

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:CW
```

class CwCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:CW
driver.source.gprf.generator.sequencer.listPy.signal.all.cw.set()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:CW
driver.source.gprf.generator.sequencer.listPy.signal.all.cw.set_with_opc()
```

No command help available

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

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

6.13.1.1.7.38 Dtone

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:DTONe
```

class DtoneCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:DTONe
driver.source.gprf.generator.sequencer.listPy.signal.all.dtone.set()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:DTONe
driver.source.gprf.generator.sequencer.listPy.signal.all.dtone.set_with_opc()
```

No command help available

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

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

6.13.1.1.7.39 Off

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:OFF
```

class OffCls

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

set() → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:OFF
driver.source.gprf.generator.sequencer.listPy.signal.all.off.set()
```

No command help available

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

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:ALL:OFF
driver.source.gprf.generator.sequencer.listPy.signal.all.off.set_with_opc()
```

No command help available

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

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

6.13.1.1.7.40 Catalog

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:CATalog:LONG
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:CATalog
```

class CatalogCls

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

class LongStruct

Structure for reading output parameters. Fields:

- Signal_Type_Index: List[int]: No parameter help available
- Signal_Type: List[str]: No parameter help available
- Arb_File_Path: List[str]: No parameter help available

get_long() → LongStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:CATalog:LONG
value: LongStruct = driver.source.gprf.generator.sequencer.listPy.signal.
↪catalog.get_long()
```

No command help available

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

get_value() → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:CATalog
value: List[str] = driver.source.gprf.generator.sequencer.listPy.signal.catalog.
↪get_value()
```

Queries all available signal types. The available types depend on the ARB file pool.

return
signal_types: Comma-separated list of strings, one string per supported signal type

6.13.1.1.7.41 Index

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:INDex
```

class IndexCls

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

class GetStruct

Response structure. Fields:

- Signal_Index: int: No parameter help available
- Signal: str: No parameter help available
- Path: str: No parameter help available

get(*row: int*) → GetStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:INDex
value: GetStruct = driver.source.gprf.generator.sequencer.listPy.signal.index.
↪get(row = 1)
```

No command help available

param row
No help available

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

set(*row: int*, *signal_index: int*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:INDex
driver.source.gprf.generator.sequencer.listPy.signal.index.set(row = 1, signal_
↪index = 1)
```

No command help available

param row
No help available

param signal_index
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.sequencer.listPy.signal.index.clone()
```

Subgroups

6.13.1.1.7.42 All

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:INDex:ALL
```

class AllCls

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

class GetStruct

Response structure. Fields:

- Signal_Index: List[int]: No parameter help available
- Signal: List[str]: No parameter help available
- Path: List[str]: No parameter help available

get() → GetStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:INDex:ALL
value: GetStruct = driver.source.gprf.generator.sequencer.listPy.signal.index.
↪all.get()
```

No command help available

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

set(*signal_index: List[int]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:INDex:ALL
driver.source.gprf.generator.sequencer.listPy.signal.index.all.set(signal_index␣
↪= [1, 2, 3])
```

No command help available

param signal_index
No help available

6.13.1.1.7.43 Range

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:RANGe
```

class RangeCls

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

get(*start_index: float*, *stop_index: float*) → List[str]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:RANGe
value: List[str] = driver.source.gprf.generator.sequencer.listPy.signal.range.
↪get(start_index = 1.0, stop_index = 1.0)
```

No command help available

param start_index
No help available

param stop_index
No help available

return
result: No help available

set(*start_index: float*, *stop_index: float*, *signal: str*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SIGNal:RANGe
driver.source.gprf.generator.sequencer.listPy.signal.range.set(start_index = 1.
↪0, stop_index = 1.0, signal = 'abc')
```

No command help available

param start_index
No help available

param stop_index
No help available

param signal
No help available

6.13.1.1.7.44 Spath

class SpathCls

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

Cloning the Group

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

Subgroups

6.13.1.1.7.45 Usage

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe
```

class UsageCls

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

get(*index: int*) → List[bool]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe
value: List[bool] = driver.source.gprf.generator.sequencer.listPy.spath.usage.
↪get(index = 1)
```

Activates or deactivates the individual RF connectors of the active connector group, for the sequencer list entry with the selected <Index>. Select the connector group via method RsCMPX_Gprf.Route.Gprf.Generator.Spath.value. Query a list of the connectors of the connector group via method RsCMPX_Gprf.Catalog.Gprf.Generator.Spath.Group.connector.

param index
Comma-separated list of values, one value per connector of the connector group. ON: activate the connector OFF: deactivate the connector

return
connectors_state: No help available

set(*index: int*, *connectors_state: List[bool]*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe
driver.source.gprf.generator.sequencer.listPy.spath.usage.set(index = 1,␣
↪connectors_state = [True, False, True])
```

Activates or deactivates the individual RF connectors of the active connector group, for the sequencer list entry with the selected <Index>. Select the connector group via method RsCMPX_Gprf.Route.Gprf.Generator.Spath.value. Query a list of the connectors of the connector group via method RsCMPX_Gprf.Catalog.Gprf.Generator.Spath.Group.connector.

param index
No help available

param connectors_state
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.sequencer.listPy.spath.usage.clone()
```

Subgroups

6.13.1.1.7.46 Bench<Bench>

RepCap Settings

```
# Range: Nr1 .. Nr20
rc = driver.source.gprf.generator.sequencer.listPy.spath.usage.bench.repcap_bench_get()
driver.source.gprf.generator.sequencer.listPy.spath.usage.bench.repcap_bench_set(repcap.
↪Bench.Nr1)
```

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe:BENCh<nr>
```

class BenchCls

Bench commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: Bench, default value after init: Bench.Nr1

get(*index: float*, *bench=Bench.Default*) → List[bool]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe:BENCh<nr>
value: List[bool] = driver.source.gprf.generator.sequencer.listPy.spath.usage.
↪bench.get(index = 1.0, bench = repcap.Bench.Default)
```

Activates or deactivates the individual RF connectors of the connector group <no>, for the sequencer list entry with the selected <Index>.

param index
No help available

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

return
enable: Comma-separated list of values, one value per connector of the connector group. ON: activate the connector OFF: deactivate the connector

set(*index: float*, *enable: List[bool]*, *bench=Bench.Default*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe:BENCh<nr>
driver.source.gprf.generator.sequencer.listPy.spath.usage.bench.set(index = 1.0,
↪ enable = [True, False, True], bench = repcap.Bench.Default)
```

Activates or deactivates the individual RF connectors of the connector group <no>, for the sequencer list entry with the selected <Index>.

param index
No help available

param enable
Comma-separated list of values, one value per connector of the connector group. ON: activate the connector OFF: deactivate the connector

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.sequencer.listPy.spath.usage.bench.clone()
```

Subgroups

6.13.1.1.7.47 Tx

class TxCls

Tx 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.gprf.generator.sequencer.listPy.spath.usage.bench.tx.clone()
```

Subgroups

6.13.1.1.7.48 Single

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe:BENCh<nr>:TX:SINGle
```

class SingleCls

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

get(*list_index: float*, *tx_index: float*, *bench=Bench.Default*) → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe:BENCh<nr>
↪:TX:SINGle
value: bool = driver.source.gprf.generator.sequencer.listPy.spath.usage.bench.
↪tx.single.get(list_index = 1.0, tx_index = 1.0, bench = repcap.Bench.Default)
```

Activates or deactivates the RF connector RF<no>.<TxIndex>+1, for the sequencer list entry with the selected <ListIndex>. Example: <no>=2 plus <TxIndex>=4 means connector RF2.5.

param list_index
No help available

param tx_index
No help available

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

return
usage: ON: activate the connector OFF: deactivate the connector

set(*list_index: float*, *tx_index: float*, *usage: bool*, *bench=Bench.Default*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SPATh:USAGe:BENCh<nr>
↪:TX:SINGle
driver.source.gprf.generator.sequencer.listPy.spath.usage.bench.tx.single.
↪set(list_index = 1.0, tx_index = 1.0, usage = False, bench = repcap.Bench.
↪Default)
```

Activates or deactivates the RF connector RF<no>.<TxIndex>+1, for the sequencer list entry with the selected <ListIndex>. Example: <no>=2 plus <TxIndex>=4 means connector RF2.5.

param list_index
No help available

param tx_index
No help available

param usage
ON: activate the connector OFF: deactivate the connector

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

6.13.1.1.7.49 SymbolRate

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SRATe
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SRATe:ALL
```

class SymbolRateCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SRATe
value: float = driver.source.gprf.generator.sequencer.listPy.symbolRate.
↪get(index = 1)
```

Queries the sample rate for the sequencer list entry with the selected <Index>.

param index
No help available

return
sample_rate: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:SRATe:ALL
value: List[float] = driver.source.gprf.generator.sequencer.listPy.symbolRate.
↪get_all()
```

Queries the sample rates for all sequencer list entries.

return
sample_rate: Comma-separated list of values, one value per list entry

6.13.1.1.7.50 Ttime

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:TTIMe
SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:TTIMe:ALL
```

class TtimeCls

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

get(*index: int*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:TTIMe
value: float = driver.source.gprf.generator.sequencer.listPy.ttime.get(index =␣
↪1)
```

Queries the transition time for the sequencer list entry with the selected <Index>.

param index
No help available

return
trans_time: No help available

get_all() → List[float]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:LIST:TTIMe:ALL
value: List[float] = driver.source.gprf.generator.sequencer.listPy.ttime.get_
↪all()
```

Queries the transition times for all sequencer list entries.

return
trans_time: Comma-separated list of values, one value per list entry

6.13.1.1.7.51 Marker

class MarkerCls

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

Cloning the Group

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

Subgroups

6.13.1.1.7.52 Delays

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:MARKer:DELays
```

class DelaysCls

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

class DelaysStruct

Response structure. Fields:

- Restart_Marker: float: No parameter help available
- Marker_2: float: No parameter help available
- Marker_3: float: No parameter help available
- Marker_4: float: No parameter help available

get() → DelaysStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:MARKer:DELays
value: DelaysStruct = driver.source.gprf.generator.sequencer.marker.delays.get()
```

Defines delay times for the ARB output trigger events relative to the marker events.

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

set(*restart_marker: float*, *marker_2: float*, *marker_3: float*, *marker_4: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:MARKer:DELays
driver.source.gprf.generator.sequencer.marker.delays.set(restart_marker = 1.0,␣
↪marker_2 = 1.0, marker_3 = 1.0, marker_4 = 1.0)
```

Defines delay times for the ARB output trigger events relative to the marker events.

param restart_marker
No help available

param marker_2
No help available

param marker_3
No help available

param marker_4
No help available

Cloning the Group

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

Subgroups

6.13.1.1.7.53 All

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:MARKer:DELays:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Restart_Marker: float: No parameter help available
- Marker_1: float: No parameter help available
- Marker_2: float: No parameter help available
- Marker_3: float: No parameter help available
- Marker_4: float: No parameter help available

get() → AllStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:MARKer:DELays:ALL
value: AllStruct = driver.source.gprf.generator.sequencer.marker.delays.all.
↪get()
```

No command help available

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

set(*restart_marker: float*, *marker_1: float*, *marker_2: float*, *marker_3: float*, *marker_4: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:MARKer:DELays:ALL
driver.source.gprf.generator.sequencer.marker.delays.all.set(restart_marker = 1.
↪0, marker_1 = 1.0, marker_2 = 1.0, marker_3 = 1.0, marker_4 = 1.0)
```

No command help available

param restart_marker
No help available

param marker_1
No help available

param marker_2
No help available

param marker_3
No help available

param marker_4
No help available

6.13.1.1.7.54 Reliability

SCPI Commands :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:RELiability
SOURce:GPRF:GENerator<Instance>:SEQuencer:RELiability:ALL
```

class ReliabilityCls

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

class AllStruct

Structure for reading output parameters. Fields:

- Reliability: int: Reliability indicator
- Reliability_Msg: str: Reason for the reliability value. Empty string '' for reliability = 0.
- Reliability_Add_Info: str: No parameter help available

get(*details: str = None*) → str

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:RELiability
value: str = driver.source.gprf.generator.sequencer.reliability.get(details =
↪'abc')
```

No command help available

Suppressed linked return values: reliability

param details
No help available

return
reliability_msg: No help available

get_all() → AllStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:RELiability:ALL
value: AllStruct = driver.source.gprf.generator.sequencer.reliability.get_all()
```

Reports if and why there are problems generating the configured signal. For possible values, see 'Reliability indicator'.

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

6.13.1.1.7.55 RfSettings

class RfSettingsCls

RfSettings 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.gprf.generator.sequencer.rfSettings.clone()
```

Subgroups

6.13.1.1.7.56 Spath

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:RFSettings:SPATh:CSET
```

class SpathCls

Spath commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_cset() → ParameterSetMode

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:RFSettings:SPATh:CSET
value: enums.ParameterSetMode = driver.source.gprf.generator.sequencer.
↪rfSettings.spath.get_cset()
```

Selects the scope of the RF connector configuration.

return

connector_set: - GLOBal: Use the same configuration for the entire sequencer list.Activate/deactivate the connectors via:CONFigure:GPRF:GENi:SPATh:USAGe orCONFigure:GPRF:GENi:SPATh:USAGe:BENChno - LIST: Configure the connectors per sequencer list entry.Activate/deactivate the connectors via:SOURce:GPRF:GENi:SEQuencer:LIST:SPATh:USAGe orSOURce:GPRF:GENi:SEQuencer:LIST:SPATh:USAGe:BENChno

set_cset(*connector_set: ParameterSetMode*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:RFSettings:SPATh:CSET
driver.source.gprf.generator.sequencer.rfSettings.spath.set_cset(connector_set␣
↪= enums.ParameterSetMode.GLOBal)
```

Selects the scope of the RF connector configuration.

param connector_set

- GLOBal: Use the same configuration for the entire sequencer list.Activate/deactivate the connectors via:CONFigure:GPRF:GENi:SPATh:USAGe orCONFigure:GPRF:GENi:SPATh:USAGe:BENChno

- LIST: Configure the connectors per sequencer list entry.Activate/deactivate the connectors via:SOURce:GPRF:GENi:SEQuencer:LIST:SPATh:USAGe or-SOURce:GPRF:GENi:SEQuencer:LIST:SPATh:USAGe:BENChno

6.13.1.1.7.57 Rmarker

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:RMARker:DELay
```

class RmarkerCls

Rmarker commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_delay() → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:RMARker:DELay
value: float = driver.source.gprf.generator.sequencer.rmarker.get_delay()
```

No command help available

return
restart_marker: No help available

set_delay(*restart_marker: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:RMARker:DELay
driver.source.gprf.generator.sequencer.rmarker.set_delay(restart_marker = 1.0)
```

No command help available

param restart_marker
No help available

6.13.1.1.7.58 State

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get() → GeneratorState

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:STATe
value: enums.GeneratorState = driver.source.gprf.generator.sequencer.state.get()
```

Turns the generator on or off.

return
generator_state: OFF: Generator switched off. PEND: State transition ongoing ON: Generator switched on, signal available. RDY: Generator switched off, sequencer list processing complete for Repetition=Single.

set(*control: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:STATe
driver.source.gprf.generator.sequencer.state.set(control = False)
```

Turns the generator on or off.

param control
Switch the generator ON or OFF.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.sequencer.state.clone()
```

Subgroups

6.13.1.1.7.59 All

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*timeout: float = None*, *target_main_state: TargetStateB = None*, *target_sync_state: TargetSyncState = None*) → List[GeneratorState]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:STATe:ALL
value: List[enums.GeneratorState] = driver.source.gprf.generator.sequencer.
↪state.all.get(timeout = 1.0, target_main_state = enums.TargetStateB.OFF,␣
↪target_sync_state = enums.TargetSyncState.ADJusted)
```

Queries the generator states. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query OFF: main state OFF RUN: main state ON STOP: main state RDY Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
all_states: No help available

6.13.1.1.7.60 Tdd

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:TDD:MODE
```

class TddCls

Tdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → bool

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:TDD:MODE
value: bool = driver.source.gprf.generator.sequencer.tdd.get_mode()
```

Enables or disables the TDD mode for a signal output path with a remote radio head (RRH) .

return
tdd_mode: No help available

set_mode(*tdd_mode: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:TDD:MODE
driver.source.gprf.generator.sequencer.tdd.set_mode(tdd_mode = False)
```

Enables or disables the TDD mode for a signal output path with a remote radio head (RRH) .

param tdd_mode
No help available

6.13.1.1.7.61 Wmarker<Marker>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.source.gprf.generator.sequencer.wmarker.repcap_marker_get()
driver.source.gprf.generator.sequencer.wmarker.repcap_marker_set(repcap.Marker.Nr1)
```

class WmarkerCls

Wmarker commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: Marker, default value after init: Marker.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.sequencer.wmarker.clone()
```

Subgroups

6.13.1.1.7.62 Delay

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:WMARker<no>:DELay
```

class DelayCls

Delay commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*marker=Marker.Default*) → float

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:WMARker<no>:DELay
value: float = driver.source.gprf.generator.sequencer.wmarker.delay.get(marker␣
↪= repcap.Marker.Default)
```

No command help available

param marker
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Wmarker')

return
waveform_marker: No help available

set(*waveform_marker: float*, *marker=Marker.Default*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:WMARker<no>:DELay
driver.source.gprf.generator.sequencer.wmarker.delay.set(waveform_marker = 1.0,␣
↪marker = repcap.Marker.Default)
```

No command help available

param waveform_marker
No help available

param marker
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Wmarker')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.sequencer.wmarker.delay.clone()
```

Subgroups

6.13.1.1.7.63 All

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:SEQuencer:WMARker:DELay:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class AllStruct

Response structure. Fields:

- Marker_1: float: No parameter help available
- Marker_2: float: No parameter help available
- Marker_3: float: No parameter help available
- Marker_4: float: No parameter help available

get() → AllStruct

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:WMARker:DELay:ALL
value: AllStruct = driver.source.gprf.generator.sequencer.wmarker.delay.all.
↪get()
```

No command help available

return
structure: for return value, see the help for AllStruct structure arguments.

set(*marker_1: float*, *marker_2: float*, *marker_3: float*, *marker_4: float*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:SEQuencer:WMARker:DELay:ALL
driver.source.gprf.generator.sequencer.wmarker.delay.all.set(marker_1 = 1.0,␣
↪marker_2 = 1.0, marker_3 = 1.0, marker_4 = 1.0)
```

No command help available

param marker_1
No help available

param marker_2
No help available

param marker_3
No help available

param marker_4
No help available

6.13.1.1.8 State

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get() → GeneratorState

```
# SCPI: SOURce:GPRF:GENerator<Instance>:STATe
value: enums.GeneratorState = driver.source.gprf.generator.state.get()
```

Turns the generator on or off.

return
generator_state: OFF: Generator switched off. PEND: State transition ongoing. ON: Generator switched on, signal available.

set(*control: bool*) → None

```
# SCPI: SOURce:GPRF:GENerator<Instance>:STATe
driver.source.gprf.generator.state.set(control = False)
```

Turns the generator on or off.

param control
Switch the generator ON or OFF.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.gprf.generator.state.clone()
```

Subgroups

6.13.1.1.8.1 All

SCPI Command :

```
SOURce:GPRF:GENerator<Instance>:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*timeout: float = None*, *target_main_state: TargetStateB = None*, *target_sync_state: TargetSyncState = None*) → List[GeneratorState]

```
# SCPI: SOURce:GPRF:GENerator<Instance>:STATe:ALL
value: List[enums.GeneratorState] = driver.source.gprf.generator.state.all.
↪get(timeout = 1.0, target_main_state = enums.TargetStateB.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the generator states. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query OFF: main state OFF RUN: main state ON Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
all_states: No help available

6.14 System

class SystemCls

System 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.system.clone()
```

Subgroups

6.14.1 Attenuation

class AttenuationCls

Attenuation 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.system.attenuation.clone()
```

Subgroups

6.14.1.1 CorrectionTable

class CorrectionTableCls

CorrectionTable commands group definition. 4 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.attenuation.correctionTable.clone()
```

Subgroups

6.14.1.1.1 All

class AllCls

All 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.attenuation.correctionTable.all.clone()
```

Subgroups

6.14.1.1.1.1 Globale

SCPI Command :

```
DELete:SYSTem:ATTenuation:CTABle:ALL:GLOBal
```

class GlobaleCls

Globale commands group definition. 1 total commands, 0 Subgroups, 1 group commands

delete() → None

```
# SCPI: DELete:SYSTem:ATTenuation:CTABle:ALL:GLOBal
driver.system.attenuation.correctionTable.all.globale.delete()
```

No command help available

delete_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: DELete:SYSTem:ATTenuation:CTABle:ALL:GLOBal
driver.system.attenuation.correctionTable.all.globale.delete_with_opc()
```

No command help available

Same as delete, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.14.1.1.1.2 Tenvironment

SCPI Command :

```
DELete:SYSTem:ATTenuation:CTABle:ALL[:TENVironment]
```

class TenvironmentCls

Tenvironment commands group definition. 1 total commands, 0 Subgroups, 1 group commands

delete() → None

```
# SCPI: DELete:SYSTem:ATTenuation:CTABle:ALL[:TENVironment]
driver.system.attenuation.correctionTable.all.tenvironment.delete()
```

No command help available

delete_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: DELete:SYSTem:ATTenuation:CTABle:ALL[:TENVironment]
driver.system.attenuation.correctionTable.all.tenvironment.delete_with_opc()
```

No command help available

Same as delete, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.14.1.1.2 Globale

SCPI Command :

```
DELete:SYSTem:ATTenuation:CTABle:GLOBal
```

class GlobaleCls

Globale commands group definition. 1 total commands, 0 Subgroups, 1 group commands

delete(*name: str*) → None

```
# SCPI: DELete:SYSTem:ATTenuation:CTABle:GLOBal
driver.system.attenuation.correctionTable.globale.delete(name = 'abc')
```

No command help available

param name
No help available

6.14.1.1.3 Tenvironment

SCPI Command :

```
DELete:SYSTem:ATTenuation:CTABle[:TENVironment]
```

class TenvironmentCls

Tenvironment commands group definition. 1 total commands, 0 Subgroups, 1 group commands

delete(*name: str*) → None

```
# SCPI: DELete:SYSTem:ATTenuation:CTABle[:TENVironment]
driver.system.attenuation.correctionTable.tenvironment.delete(name = 'abc')
```

No command help available

param name
No help available

6.14.2 Date

class DateCls

Date 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.date.clone()
```

Subgroups

6.14.2.1 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

6.14.3 Time

SCPI Commands :

```
SYSTem:TIME:SOURce
SYSTem:TIME:NTP
```

class TimeCls

Time commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_ntp() → str

```
# SCPI: SYSTem:TIME:NTP
value: str = driver.system.time.get_ntp()
```

No command help available

return
time_server: No help available

get_source() → TimeSource

```
# SCPI: SYSTem:TIME:SOURce
value: enums.TimeSource = driver.system.time.get_source()
```

No command help available

return
time_source: No help available

set_ntp(*time_server: str*) → None

```
# SCPI: SYSTem:TIME:NTP
driver.system.time.set_ntp(time_server = 'abc')
```

No command help available

param time_server
No help available

set_source(*time_source: TimeSource*) → None

```
# SCPI: SYSTem:TIME:SOURce
driver.system.time.set_source(time_source = enums.TimeSource.MANual)
```

No command help available

param time_source
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.time.clone()
```

Subgroups

6.14.3.1 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

6.15 Tenvironment

class TenvironmentCls

Tenvironment 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.tenvironment.clone()
```

Subgroups

6.15.1 Spath

SCPI Command :

```
DELete:TENVironment:SPATh
```

class SpathCls

Spath commands group definition. 1 total commands, 0 Subgroups, 1 group commands

delete(*name_signal_path: str*) → None

```
# SCPI: DELete:TENVironment:SPATh
driver.tenvironment.spath.delete(name_signal_path = 'abc')
```

No command help available

param name_signal_path
No help available

6.16 Trigger

class TriggerCls

Trigger commands group definition. 109 total commands, 14 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.clone()
```

Subgroups

6.16.1 Bluetooth

class BluetoothCls

Bluetooth 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.trigger.bluetooth.clone()
```

Subgroups

6.16.1.1 Measurement

class MeasurementCls

Measurement commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.bluetooth.measurement.clone()
```

Subgroups

6.16.1.1.1 BhRate

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:BHRate:SOURce
```

class BhRateCls

BhRate commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:BHRate:SOURce
value: str = driver.trigger.bluetooth.measurement.bhRate.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:BHRate:SOURce
driver.trigger.bluetooth.measurement.bhRate.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.bluetooth.measurement.bhRate.clone()
```

Subgroups

6.16.1.1.1.1 Catalog

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:BHRate:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:BHRate:CATalog:SOURce
value: List[str] = driver.trigger.bluetooth.measurement.bhRate.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.1.1.2 Hdr

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:HDR:SOURce
```

class HdrCls

Hdr commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:HDR:SOURce
value: str = driver.trigger.bluetooth.measurement.hdr.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:HDR:SOURce
driver.trigger.bluetooth.measurement.hdr.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.bluetooth.measurement.hdr.clone()
```

Subgroups

6.16.1.1.2.1 Catalog

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:HDR:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:HDR:CATalog:SOURce
value: List[str] = driver.trigger.bluetooth.measurement.hdr.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.1.1.3 Hdrp

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:HDRP:SOURce
```

class HdrpCls

Hdrp commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:HDRP:SOURce
value: str = driver.trigger.bluetooth.measurement.hdrp.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:HDRP:SOURce
driver.trigger.bluetooth.measurement.hdrp.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.bluetooth.measurement.hdrp.clone()
```

Subgroups

6.16.1.1.3.1 Catalog

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:HDRP:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:HDRP:CATalog:SOURce
value: List[str] = driver.trigger.bluetooth.measurement.hdrp.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.1.1.4 MultiEval

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.bluetooth.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.bluetooth.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.bluetooth.measurement.multiEval.clone()
```

Subgroups

6.16.1.1.4.1 Catalog

SCPI Command :

```
TRIGger:BLUetooth:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:BLUetooth:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.bluetooth.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.2 Cdma

class CdmaCls

Cdma 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.trigger.cdma.clone()
```

Subgroups

6.16.2.1 Measurement

class MeasurementCls

Measurement 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.trigger.cdma.measurement.clone()
```

Subgroups

6.16.2.1.1 MultiEval

SCPI Command :

```
TRIGger:CDMA:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:CDMA:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.cdma.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:CDMA:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.cdma.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.cdma.measurement.multiEval.clone()
```

Subgroups

6.16.2.1.1.1 Catalog

SCPI Command :

```
TRIGger:CDMA:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:CDMA:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.cdma.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.3 Gprf

class GprfCls

Gprf commands group definition. 57 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.clone()
```

Subgroups

6.16.3.1 Generator

SCPI Command :

```
TRIGger:GPRF:GENerator<Instance>:TOUT
```

class GeneratorCls

Generator commands group definition. 16 total commands, 2 Subgroups, 1 group commands

get_timeout() → float

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:TOUT
value: float or bool = driver.trigger.gprf.generator.get_timeout()
```

Sets a time after which the generator must have received a trigger event, after it has been started.

return
timeout: (float or boolean) No help available

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:TOUT
driver.trigger.gprf.generator.set_timeout(timeout = 1.0)
```

Sets a time after which the generator must have received a trigger event, after it has been started.

param timeout
(float or boolean) No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.generator.clone()
```

Subgroups

6.16.3.1.1 Arb

SCPI Commands :

```
TRIGger:GPRF:GENerator<Instance>:ARB:DELay
TRIGger:GPRF:GENerator<Instance>:ARB:SLOPe
TRIGger:GPRF:GENerator<Instance>:ARB:RETRigger
TRIGger:GPRF:GENerator<Instance>:ARB:AUTostart
TRIGger:GPRF:GENerator<Instance>[:ARB]:SOURce
```

class ArbCls

Arb commands group definition. 9 total commands, 3 Subgroups, 5 group commands

get_autostart() → bool

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:AUTostart
value: bool = driver.trigger.gprf.generator.arb.get_autostart()
```

No command help available

return
autostart: No help available

get_delay() → float

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:DELay
value: float = driver.trigger.gprf.generator.arb.get_delay()
```

Sets the trigger delay.

return
delay: No help available

get_retrigger() → bool

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:RETRigger
value: bool = driver.trigger.gprf.generator.arb.get_retrigger()
```

No command help available

return
retrigger: No help available

get_slope() → SignalSlope

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:SLOPe
value: enums.SignalSlope = driver.trigger.gprf.generator.arb.get_slope()
```

No command help available

return
slope: No help available

get_source() → str

```
# SCPI: TRIGger:GPRF:GENerator<Instance>[:ARB]:SOURce
value: str = driver.trigger.gprf.generator.arb.get_source()
```

Selects the source of the trigger events. The supported values depend on the installed options. You can query a list of all supported values via method RsCMPX_Gprf.Trigger.Gprf.Generator.Arb.Catalog.source.

return
trigger: No help available

set_autostart(*autostart: bool*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:AUTostart
driver.trigger.gprf.generator.arb.set_autostart(autostart = False)
```

No command help available

param autostart
No help available

set_delay(*delay: float*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:DELay
driver.trigger.gprf.generator.arb.set_delay(delay = 1.0)
```

Sets the trigger delay.

param delay
No help available

set_retrigger(*retrigger: bool*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:RETRigger
driver.trigger.gprf.generator.arb.set_retrigger(retrigger = False)
```

No command help available

param retrigger
No help available

set_slope(*slope: SignalSlope*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:SLOPe
driver.trigger.gprf.generator.arb.set_slope(slope = enums.SignalSlope.FEDGe)
```

No command help available

param slope
No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>[:ARB]:SOURce
driver.trigger.gprf.generator.arb.set_source(trigger = 'abc')
```

Selects the source of the trigger events. The supported values depend on the installed options. You can query a list of all supported values via method RsCMPX_Gprf.Trigger.Gprf.Generator.Arb.Catalog.source.

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.generator.arb.clone()
```

Subgroups

6.16.3.1.1.1 Catalog

SCPI Command :

```
TRIGger:GPRF:GENerator<Instance>[:ARB]:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:GPRF:GENerator<Instance>[:ARB]:CATalog:SOURce
value: List[str] = driver.trigger.gprf.generator.arb.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCMPX_Gprf.Trigger.Gprf.Generator.Arb.source.

return
trigger: Comma-separated list of all supported values. Each value is represented as a string.

6.16.3.1.1.2 Manual

class ManualCls

Manual 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.gprf.generator.arb.manual.clone()
```

Subgroups

6.16.3.1.1.3 Execute

SCPI Command :

```
TRIGger:GPRF:GENerator<Instance>:ARB:MANual:EXECute
```

class ExecuteCls

Execute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:MANual:EXECute
driver.trigger.gprf.generator.arb.manual.execute.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:MANual:EXECute
driver.trigger.gprf.generator.arb.manual.execute.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.16.3.1.1.4 Segments

SCPI Command :

```
TRIGger:GPRF:GENerator<Instance>:ARB:SEGMents:MODE
```

class SegmentsCls

Segments commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_mode() → ArbSegmentsMode

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:SEGMents:MODE
value: enums.ArbSegmentsMode = driver.trigger.gprf.generator.arb.segments.get_
↪mode()
```

No command help available

return
mode: No help available

set_mode(*mode: ArbSegmentsMode*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:SEGMents:MODE
driver.trigger.gprf.generator.arb.segments.set_mode(mode = enums.
↪ArbSegmentsMode.AUTO)
```

No command help available

param mode
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.generator.arb.segments.clone()
```

Subgroups

6.16.3.1.1.5 Manual

class ManualCls

Manual 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.gprf.generator.arb.segments.manual.clone()
```

Subgroups

6.16.3.1.1.6 Execute

SCPI Command :

```
TRIGger:GPRF:GENerator<Instance>:ARB:SEGMents:MANual:EXECute
```

class ExecuteCls

Execute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:SEGMents:MANual:EXECute
driver.trigger.gprf.generator.arb.segments.manual.execute.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:ARB:SEGMents:MANual:EXECute
driver.trigger.gprf.generator.arb.segments.manual.execute.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.16.3.1.2 Sequencer

SCPI Command :

```
TRIGger:GPRF:GENerator<Instance>:SEQuencer:TOUT
```

class SequencerCls

Sequencer commands group definition. 6 total commands, 3 Subgroups, 1 group commands

get_timeout() → float

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:TOUT
value: float or bool = driver.trigger.gprf.generator.sequencer.get_timeout()
```

Sets a timeout for waiting for a trigger event for List Increment = MEASUREMENT and TRIGGER.

return
timeout: (float or boolean) No help available

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:TOUT
driver.trigger.gprf.generator.sequencer.set_timeout(timeout = 1.0)
```

Sets a timeout for waiting for a trigger event for List Increment = MEASUREMENT and TRIGGER.

param timeout
(float or boolean) No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.generator.sequencer.clone()
```

Subgroups

6.16.3.1.2.1 IsMeas

SCPI Commands :

```
TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISMeas:CATalog
TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISMeas:SOURce
```

class IsMeasCls

IsMeas commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_catalog() → List[str]

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISMeas:CATalog
value: List[str] = driver.trigger.gprf.generator.sequencer.isMeas.get_catalog()
```

Queries all available measurement source strings.

return
trigger: Comma-separated list of strings. Each string represents a supported source.

get_source() → List[str]

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISMeas:SOURce
value: List[str] = driver.trigger.gprf.generator.sequencer.isMeas.get_source()
```

Selects a measurement for triggering sequencer list incrementations. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Trigger.Gprf.Generator.Sequencer.IsMeas.catalog.

return
trigger: No help available

set_source(*trigger: List[str]*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISMeas:SOURce
driver.trigger.gprf.generator.sequencer.isMeas.set_source(trigger = ['abc1',
↪'abc2', 'abc3'])
```

Selects a measurement for triggering sequencer list incrementations. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Trigger.Gprf.Generator.Sequencer.IsMeas.catalog.

param trigger
No help available

6.16.3.1.2.2 IsTrigger

SCPI Commands :

```
TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISTRigger:CATalog
TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISTRigger:SOURce
```

class IsTriggerCls

IsTrigger commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_catalog() → List[str]

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISTRigger:CATalog
value: List[str] = driver.trigger.gprf.generator.sequencer.isTrigger.get_
↪catalog()
```

Queries all available trigger source strings.

return
trigger: Comma-separated list of strings. Each string represents a supported source.

get_source() → str

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISTRigger:SOURce
value: str = driver.trigger.gprf.generator.sequencer.isTrigger.get_source()
```

Selects a trigger source for triggering sequencer list incrementations. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Trigger.Gprf.Generator.Sequencer.IsTrigger.catalog.

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:ISTRigger:SOURce
driver.trigger.gprf.generator.sequencer.isTrigger.set_source(trigger = 'abc')
```

Selects a trigger source for triggering sequencer list incrementations. A complete list of all supported strings can be queried using method RsCMPX_Gprf.Trigger.Gprf.Generator.Sequencer.IsTrigger.catalog.

param trigger
No help available

6.16.3.1.2.3 Manual

class ManualCls

Manual 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.gprf.generator.sequencer.manual.clone()
```

Subgroups

6.16.3.1.2.4 Execute

SCPI Command :

```
TRIGger:GPRF:GENerator<Instance>:SEQuencer:MANual:EXECute
```

class ExecuteCls

Execute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:MANual:EXECute
driver.trigger.gprf.generator.sequencer.manual.execute.set()
```

Triggers the transition to the next sequencer list entry manually.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: TRIGger:GPRF:GENerator<Instance>:SEQuencer:MANual:EXECute
driver.trigger.gprf.generator.sequencer.manual.execute.set_with_opc()
```

Triggers the transition to the next sequencer list entry manually.

Same as set, but waits for the operation to complete before continuing further. Use the RsCMPX_Gprf.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.16.3.2 Measurement

class MeasurementCls

Measurement commands group definition. 41 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.measurement.clone()
```

Subgroups

6.16.3.2.1 FftSpecAn

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:MGAP
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OFFSet
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:THReshold
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SLOPe
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
```

class FftSpecAnCls

FftSpecAn commands group definition. 9 total commands, 2 Subgroups, 7 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:MGAP
value: float = driver.trigger.gprf.measurement.fftSpecAn.get_mgap()
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

return
minimum_gap: No help available

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OFFSet
value: float = driver.trigger.gprf.measurement.fftSpecAn.get_offset()
```

Defines the trigger offset for the trigger offset mode FIXed. The trigger offset defines the center of the measurement interval relative to the trigger event.

return
offset: No help available

get_omode() → OffsetMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
value: enums.OffsetMode = driver.trigger.gprf.measurement.fftSpecAn.get_omode()
```

Selects the trigger offset mode.

return
offset_mode: No help available

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprf.measurement.fftSpecAn.get_
↪slope()
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

return
event: REDGe: rising edge FEDGe: falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
value: str = driver.trigger.gprf.measurement.fftSpecAn.get_source()
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

return
trigger: No help available

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:THReshold
value: float = driver.trigger.gprf.measurement.fftSpecAn.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: No help available

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
value: float or bool = driver.trigger.gprf.measurement.fftSpecAn.get_timeout()
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

return
timeout: (float or boolean) No help available

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:MGAP
driver.trigger.gprf.measurement.fftSpecAn.set_mgap(minimum_gap = 1.0)
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

param minimum_gap
No help available

set_offset(*offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OFFSet
driver.trigger.gprf.measurement.fftSpecAn.set_offset(offset = 1.0)
```

Defines the trigger offset for the trigger offset mode FIXed. The trigger offset defines the center of the measurement interval relative to the trigger event.

param offset
No help available

set_omode(*offset_mode: OffsetMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
driver.trigger.gprf.measurement.fftSpecAn.set_omode(offset_mode = enums.
↪OffsetMode.FIXed)
```

Selects the trigger offset mode.

param offset_mode
No help available

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SLOPe
driver.trigger.gprf.measurement.fftSpecAn.set_slope(event = enums.
↪SignalSlopeExt.FALLing)
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

param event
REDGe: rising edge FEDGe: falling edge

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
driver.trigger.gprf.measurement.fftSpecAn.set_source(trigger = 'abc')
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

param trigger
'IF Power': IF power trigger 'Free Run': free run (untriggered)

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:THReshold
driver.trigger.gprf.measurement.fftSpecAn.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
No help available

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
driver.trigger.gprf.measurement.fftSpecAn.set_timeout(timeout = 1.0)
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

param timeout
(float or boolean) No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.measurement.fftSpecAn.clone()
```

Subgroups

6.16.3.2.1.1 Catalog

SCPI Command :

```
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:CATalog:SOURce
value: List[str] = driver.trigger.gprf.measurement.fftSpecAn.catalog.get_
↪source()
```

Lists all trigger source values that can be set using method RsCMPX_Gprf.Trigger.Gprf.Measurement.FftSpecAn.source.

return
trigger: Comma-separated list of all supported values. Each value is represented as a string.

6.16.3.2.1.2 OsStop

SCPI Command :

```
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OSSTop
```

class OsStopCls

OsStop commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class OsStopStruct

Response structure. Fields:

- Offset_Start: float: No parameter help available
- Offset_Stop: float: No parameter help available

get() → OsStopStruct

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OSSTop
value: OsStopStruct = driver.trigger.gprf.measurement.fftSpecAn.osStop.get()
```

Defines the start and stop values for the trigger-offset mode VARiable. The start value must be smaller than the stop value.

return
structure: for return value, see the help for OsStopStruct structure arguments.

set(*offset_start: float*, *offset_stop: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OSSTop
driver.trigger.gprf.measurement.fftSpecAn.osStop.set(offset_start = 1.0, offset_
↪stop = 1.0)
```

Defines the start and stop values for the trigger-offset mode VARiable. The start value must be smaller than the stop value.

param offset_start
No help available

param offset_stop
No help available

6.16.3.2.2 IqRecorder

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:MGAP
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTHreshold
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTime
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SLOPe
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
```

class IqRecorderCls

IqRecorder commands group definition. 9 total commands, 1 Subgroups, 8 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:MGAP
value: float = driver.trigger.gprf.measurement.iqRecorder.get_mgap()
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated. The I/Q recorder always runs in single-shot mode. Therefore it is controlled by a single trigger event. The minimum trigger gap condition is valid between the start of the measurement and the first trigger event.

return
minimum_gap: No help available

get_offset() → int

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
value: int = driver.trigger.gprf.measurement.iqRecorder.get_offset()
```

Defines a delay time for triggered measurements. The trigger offset delays the start of the measurement relative to the trigger event.

return
trigger_offset: Trigger offset in samples.

get_pc_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTHreshold
value: float = driver.trigger.gprf.measurement.iqRecorder.get_pc_threshold()
```

Defines the minimum absolute phase change required to generate a trigger event for the phase change trigger source.

return
phase_chg_thres: No help available

get_pc_time() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTime
value: float or bool = driver.trigger.gprf.measurement.iqRecorder.get_pc_time()
```

Defines the time interval during which a phase change must occur to generate a trigger event for the phase change trigger source.

return
phase_chg_time: (float or boolean) No help available

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprf.measurement.iqRecorder.get_
↪slope()
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

return
event: REDGe: rising edge FEDGe: falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
value: str = driver.trigger.gprf.measurement.iqRecorder.get_source()
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

return
trigger: No help available

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
value: float = driver.trigger.gprf.measurement.iqRecorder.get_threshold()
```

Defines the trigger threshold for power trigger sources. Signals below the threshold are not evaluated for the phase change trigger source.

return
threshold: No help available

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
value: float or bool = driver.trigger.gprf.measurement.iqRecorder.get_timeout()
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

return
timeout: (float or boolean) No help available

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:MGAP
driver.trigger.gprf.measurement.iqRecorder.set_mgap(minimum_gap = 1.0)
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated. The I/Q recorder always runs in single-shot mode. Therefore it is controlled by a single trigger event. The minimum trigger gap condition is valid between the start of the measurement and the first trigger event.

param minimum_gap
No help available

set_offset(*trigger_offset: int*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
driver.trigger.gprf.measurement.iqRecorder.set_offset(trigger_offset = 1)
```

Defines a delay time for triggered measurements. The trigger offset delays the start of the measurement relative to the trigger event.

param trigger_offset
Trigger offset in samples.

set_pc_threshold(*phase_chg_thres: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTHreshold
driver.trigger.gprf.measurement.iqRecorder.set_pc_threshold(phase_chg_thres = 1.
↪0)
```

Defines the minimum absolute phase change required to generate a trigger event for the phase change trigger source.

param phase_chg_thres
No help available

set_pc_time(*phase_chg_time: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTime
driver.trigger.gprf.measurement.iqRecorder.set_pc_time(phase_chg_time = 1.0)
```

Defines the time interval during which a phase change must occur to generate a trigger event for the phase change trigger source.

param phase_chg_time
(float or boolean) No help available

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SLOPe
driver.trigger.gprf.measurement.iqRecorder.set_slope(event = enums.
↪SignalSlopeExt.FALLing)
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

param event
REDGe: rising edge FEDGe: falling edge

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
driver.trigger.gprf.measurement.iqRecorder.set_source(trigger = 'abc')
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

param trigger
'IF Power': IF power trigger 'Free Run': free run (untriggered) 'Phase Change': phase change trigger

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
driver.trigger.gprf.measurement.iqRecorder.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources. Signals below the threshold are not evaluated for the phase change trigger source.

param threshold
No help available

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
driver.trigger.gprf.measurement.iqRecorder.set_timeout(timeout = 1.0)
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

param timeout
(float or boolean) No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.measurement.iqRecorder.clone()
```

Subgroups

6.16.3.2.2.1 Catalog

SCPI Command :

```
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:CATalog:SOURce
value: List[str] = driver.trigger.gprf.measurement.iqRecorder.catalog.get_
↪source()
```

Lists all trigger source values that can be set using method RsCMPX_Gprf.Trigger.Gprf.Measurement.IqRecorder.source.

return
trigger: Comma-separated list of all supported values. Each value is represented as a string.

6.16.3.2.3 IqVsSlot

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MGAP
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:OFFSet
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:TOUT
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:THReshold
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SLOPe
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MODE
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SOURce
```

class IqVsSlotCls

IqVsSlot commands group definition. 8 total commands, 1 Subgroups, 7 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MGAP
value: float = driver.trigger.gprf.measurement.iqVsSlot.get_mgap()
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

return
minimum_gap: No help available

get_mode() → TriggerSequenceMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MODE
value: enums.TriggerSequenceMode = driver.trigger.gprf.measurement.iqVsSlot.get_
↪mode()
```

Selects the measurement sequence that is triggered by each single trigger event. This setting is not valid for free run measurements.

return
mode: ONCE: Trigger Once PRESelect: Retrigger Preselect

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:OFFSet
value: float = driver.trigger.gprf.measurement.iqVsSlot.get_offset()
```

Defines a delay time for triggered measurements. The trigger offset delays the start of the measurement relative to the trigger event.

return
offset: No help available

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprf.measurement.iqVsSlot.get_
↪slope()
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

return
event: REDGe: Rising edge FEDGe: Falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SOURce
value: str = driver.trigger.gprf.measurement.iqVsSlot.get_source()
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

return
trigger: No help available

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:THReshold
value: float = driver.trigger.gprf.measurement.iqVsSlot.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: No help available

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:TOUT
value: float or bool = driver.trigger.gprf.measurement.iqVsSlot.get_timeout()
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

return
timeout: (float or boolean) No help available

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MGAP
driver.trigger.gprf.measurement.iqVsSlot.set_mgap(minimum_gap = 1.0)
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

param minimum_gap
No help available

set_mode(*mode: TriggerSequenceMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MODE
driver.trigger.gprf.measurement.iqVsSlot.set_mode(mode = enums.
↪TriggerSequenceMode.ONCE)
```

Selects the measurement sequence that is triggered by each single trigger event. This setting is not valid for free run measurements.

param mode
ONCE: Trigger Once PRESelect: Retrigger Preselect

set_offset(*offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:OFFSet
driver.trigger.gprf.measurement.iqVsSlot.set_offset(offset = 1.0)
```

Defines a delay time for triggered measurements. The trigger offset delays the start of the measurement relative to the trigger event.

param offset
No help available

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SLOPe
driver.trigger.gprf.measurement.iqVsSlot.set_slope(event = enums.SignalSlopeExt.
↪FALLing)
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

param event
REDGe: Rising edge FEDGe: Falling edge

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SOURce
driver.trigger.gprf.measurement.iqVsSlot.set_source(trigger = 'abc')
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

param trigger
'IF Power': IF power trigger 'Free Run': free run (untriggered)

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:THReshold
driver.trigger.gprf.measurement.iqVsSlot.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
No help available

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:TOUT
driver.trigger.gprf.measurement.iqVsSlot.set_timeout(timeout = 1.0)
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

param timeout
(float or boolean) No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.measurement.iqVsSlot.clone()
```

Subgroups

6.16.3.2.3.1 Catalog

SCPI Command :

```
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:CATalog:SOURce
value: List[str] = driver.trigger.gprf.measurement.iqVsSlot.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCMPX_Gprf.Trigger.Gprf.Measurement.IqVsSlot.source.

return
trigger: Comma-separated list of all supported values. Each value is represented as a string.

6.16.3.2.4 Power

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:POWer:MGAP
TRIGger:GPRF:MEASurement<Instance>:POWer:TOUT
TRIGger:GPRF:MEASurement<Instance>:POWer:OFFSet
TRIGger:GPRF:MEASurement<Instance>:POWer:MODE
TRIGger:GPRF:MEASurement<Instance>:POWer:THReshold
TRIGger:GPRF:MEASurement<Instance>:POWer:SLOPe
TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
```

class PowerCls

Power commands group definition. 10 total commands, 2 Subgroups, 7 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MGAP
value: float = driver.trigger.gprf.measurement.power.get_mgap()
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

return
minimum_gap: No help available

get_mode() → TriggerPowerMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MODE
value: enums.TriggerPowerMode = driver.trigger.gprf.measurement.power.get_mode()
```

Selects the measurement sequence that is triggered by each single trigger event. This setting is not valid for free run measurements.

return
mode: ONCE: Trigger Once SWEep: Retrigger Sweep ALL: Retrigger All PRESelect: Retrigger Preselect

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:OFFSet
value: float = driver.trigger.gprf.measurement.power.get_offset()
```

Defines a delay time for triggered measurements. The trigger offset delays the start of the measurement relative to the trigger event.

return
offset: No help available

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprf.measurement.power.get_slope()
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

return
event: REDGe: rising edge FEDGe: falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
value: str = driver.trigger.gprf.measurement.power.get_source()
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

return
trigger: No help available

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:THReshold
value: float = driver.trigger.gprf.measurement.power.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: No help available

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:TOUT
value: float or bool = driver.trigger.gprf.measurement.power.get_timeout()
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

return
timeout: (float or boolean) No help available

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MGAP
driver.trigger.gprf.measurement.power.set_mgap(minimum_gap = 1.0)
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

param minimum_gap
No help available

set_mode(*mode: TriggerPowerMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MODE
driver.trigger.gprf.measurement.power.set_mode(mode = enums.TriggerPowerMode.
↪ALL)
```

Selects the measurement sequence that is triggered by each single trigger event. This setting is not valid for free run measurements.

param mode
ONCE: Trigger Once SWEep: Retrigger Sweep ALL: Retrigger All PRESelect: Retrigger Preselect

set_offset(*offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:OFFSet
driver.trigger.gprf.measurement.power.set_offset(offset = 1.0)
```

Defines a delay time for triggered measurements. The trigger offset delays the start of the measurement relative to the trigger event.

param offset
No help available

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SLOPe
driver.trigger.gprf.measurement.power.set_slope(event = enums.SignalSlopeExt.
↪FALLing)
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

param event
REDGe: rising edge FEDGe: falling edge

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
driver.trigger.gprf.measurement.power.set_source(trigger = 'abc')
```

Selects the source of the trigger events. Some values are always available. They are listed below. Depending on the installed options, additional values are available. You can query a list of all supported values via TRIGger:... :CATalog:SOURce?.

param trigger
'IF Power': IF power trigger 'Free Run': free run (untriggered)

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:THReshold
driver.trigger.gprf.measurement.power.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
No help available

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:TOUT
driver.trigger.gprf.measurement.power.set_timeout(timeout = 1.0)
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

param timeout
(float or boolean) No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprf.measurement.power.clone()
```

Subgroups

6.16.3.2.4.1 Catalog

SCPI Command :

```
TRIGger:GPRF:MEASurement<Instance>:POWer:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:CATalog:SOURce
value: List[str] = driver.trigger.gprf.measurement.power.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCMPX_Gprf.Trigger.Gprf.Measurement.Power.source.

return
trigger: Comma-separated list of all supported values. Each value is represented as a string.

6.16.3.2.4.2 ParameterSetList

class ParameterSetListCls

ParameterSetList 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.trigger.gprf.measurement.power.parameterSetList.clone()
```

Subgroups

6.16.3.2.4.3 Offset

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet
TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet:ALL
```

class OffsetCls

Offset commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get(*index: int*) → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet
value: float = driver.trigger.gprf.measurement.power.parameterSetList.offset.
↪get(index = 1)
```

Defines a delay time relative to the trigger event for the parameter set <Index>.

param index
No help available

return
trigger_offset: No help available

`get_all`() → List[float]

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet:ALL
value: List[float] = driver.trigger.gprf.measurement.power.parameterSetList.
↪offset.get_all()
```

Defines a delay time relative to the trigger event for all parameter sets.

return
trigger_offset: Comma-separated list of 32 offsets, for parameter set 0 to 31.

`set`(*index: int*, *trigger_offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet
driver.trigger.gprf.measurement.power.parameterSetList.offset.set(index = 1,␣
↪trigger_offset = 1.0)
```

Defines a delay time relative to the trigger event for the parameter set <Index>.

param index
No help available

param trigger_offset
No help available

`set_all`(*trigger_offset: List[float]*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet:ALL
driver.trigger.gprf.measurement.power.parameterSetList.offset.set_all(trigger_
↪offset = [1.1, 2.2, 3.3])
```

Defines a delay time relative to the trigger event for all parameter sets.

param trigger_offset
Comma-separated list of 32 offsets, for parameter set 0 to 31.

6.16.3.2.5 Spectrum

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SLOPe
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:MGAP
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:OFFSet
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:TOUT
```

`class SpectrumCls`

Spectrum commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:MGAP
value: float = driver.trigger.gprf.measurement.spectrum.get_mgap()
```

No command help available

return
minimum_gap: No help available

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:OFFSet
value: float = driver.trigger.gprf.measurement.spectrum.get_offset()
```

No command help available

return
trigger_offset: No help available

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprf.measurement.spectrum.get_
↪slope()
```

No command help available

return
slope: No help available

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
value: float = driver.trigger.gprf.measurement.spectrum.get_threshold()
```

No command help available

return
threshold: No help available

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:TOUT
value: float or bool = driver.trigger.gprf.measurement.spectrum.get_timeout()
```

No command help available

return
trigger_timeout: (float or boolean) No help available

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:MGAP
driver.trigger.gprf.measurement.spectrum.set_mgap(minimum_gap = 1.0)
```

No command help available

param minimum_gap
No help available

set_offset(*trigger_offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:OFFSet
driver.trigger.gprf.measurement.spectrum.set_offset(trigger_offset = 1.0)
```

No command help available

param trigger_offset
No help available

set_slope(*slope: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SLOPe
driver.trigger.gprf.measurement.spectrum.set_slope(slope = enums.SignalSlopeExt.
↪FALLing)
```

No command help available

param slope
No help available

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
driver.trigger.gprf.measurement.spectrum.set_threshold(threshold = 1.0)
```

No command help available

param threshold
No help available

set_timeout(*trigger_timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:TOUT
driver.trigger.gprf.measurement.spectrum.set_timeout(trigger_timeout = 1.0)
```

No command help available

param trigger_timeout
(float or boolean) No help available

6.16.4 Gsm

class GsmCls

Gsm 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.trigger.gsm.clone()
```

Subgroups

6.16.4.1 Measurement

class MeasurementCls

Measurement 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.trigger.gsm.measurement.clone()
```

Subgroups

6.16.4.1.1 MultiEval

SCPI Command :

```
TRIGger:GSM:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:GSM:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.gsm.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:GSM:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.gsm.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gsm.measurement.multiEval.clone()
```

Subgroups

6.16.4.1.1.1 Catalog

SCPI Command :

```
TRIGger:GSM:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:GSM:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.gsm.measurement.multiEval.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.5 Lte

class LteCls

Lte commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lte.clone()
```

Subgroups

6.16.5.1 Measurement

class MeasurementCls

Measurement commands group definition. 6 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lte.measurement.clone()
```

Subgroups

6.16.5.1.1 MultiEval

SCPI Command :

```
TRIGger:LTE:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.lte.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.lte.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lte.measurement.multiEval.clone()
```

Subgroups

6.16.5.1.1.1 Catalog

SCPI Command :

```
TRIGger:LTE:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.lte.measurement.multiEval.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.5.1.2 Prach

SCPI Command :

```
TRIGger:LTE:MEASurement<Instance>:PRACh:SOURce
```

class PrachCls

Prach commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:PRACh:SOURce
value: str = driver.trigger.lte.measurement.prach.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:PRACh:SOURce
driver.trigger.lte.measurement.prach.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lte.measurement.prach.clone()
```

Subgroups

6.16.5.1.2.1 Catalog

SCPI Command :

```
TRIGger:LTE:MEASurement<Instance>:PRACh:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:PRACh:CATalog:SOURce
value: List[str] = driver.trigger.lte.measurement.prach.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.5.1.3 Srs

SCPI Command :

```
TRIGger:LTE:MEASurement<Instance>:SRS:SOURce
```

class SrsCls

Srs commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:SRS:SOURce
value: str = driver.trigger.lte.measurement.srs.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:SRS:SOURce
driver.trigger.lte.measurement.srs.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lte.measurement.srs.clone()
```

Subgroups

6.16.5.1.3.1 Catalog

SCPI Command :

```
TRIGger:LTE:MEASurement<Instance>:SRS:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:LTE:MEASurement<Instance>:SRS:CATalog:SOURce
value: List[str] = driver.trigger.lte.measurement.srs.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.6 LteDl

class LteDlCls

LteDl 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.trigger.lteDl.clone()
```

Subgroups

6.16.6.1 Measurement

class MeasurementCls

Measurement 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.trigger.lteDl.measurement.clone()
```

Subgroups

6.16.6.1.1 MultiEval

SCPI Command :

```
TRIGger:LTEDl:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:LTEDl:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.lteDl.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:LTEDl:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.lteDl.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.lteDl.measurement.multiEval.clone()
```

Subgroups

6.16.6.1.1.1 Catalog

SCPI Command :

```
TRIGger:LTEDl:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:LTEDl:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.lteDl.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.7 Niot

class NiotCls

Niot 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.trigger.niot.clone()
```

Subgroups

6.16.7.1 Measurement

class MeasurementCls

Measurement 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.trigger.niot.measurement.clone()
```

Subgroups

6.16.7.1.1 MultiEval

SCPI Command :

```
TRIGger:NIOT:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.niot.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.niot.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.niot.measurement.multiEval.clone()
```

Subgroups

6.16.7.1.1.1 Catalog

SCPI Command :

```
TRIGger:NIOT:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.niot.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.7.1.2 Prach

SCPI Command :

```
TRIGger:NIOT:MEASurement<Instance>:PRACh:SOURce
```

class PrachCls

Prach commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:SOURce
value: str = driver.trigger.niot.measurement.prach.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:SOURce
driver.trigger.niot.measurement.prach.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.niot.measurement.prach.clone()
```

Subgroups

6.16.7.1.2.1 Catalog

SCPI Command :

```
TRIGger:NIOT:MEASurement<Instance>:PRACh:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:CATalog:SOURce
value: List[str] = driver.trigger.niot.measurement.prach.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.8 NrDl

class NrDlCls

NrDl 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.trigger.nrDl.clone()
```

Subgroups

6.16.8.1 Measurement

class MeasurementCls

Measurement 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.trigger.nrDl.measurement.clone()
```

Subgroups

6.16.8.1.1 MultiEval

SCPI Command :

```
TRIGger:NRDL:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NRDL:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.nrDl.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NRDL:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.nrDl.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrDl.measurement.multiEval.clone()
```

Subgroups

6.16.8.1.1.1 Catalog

SCPI Command :

```
TRIGger:NRDL:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NRDL:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.nrDl.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.9 NrMmw

class NrMmwCls

NrMmw 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.trigger.nrMmw.clone()
```

Subgroups

6.16.9.1 Measurement

class MeasurementCls

Measurement 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.trigger.nrMmw.measurement.clone()
```

Subgroups

6.16.9.1.1 MultiEval

SCPI Command :

```
TRIGger:NRMMw:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NRMMw:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.nrMmw.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NRMMw:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.nrMmw.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrMmw.measurement.multiEval.clone()
```

Subgroups

6.16.9.1.1.1 Catalog

SCPI Command :

```
TRIGger:NRMMw:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NRMMw:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.nrMmw.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.9.1.2 Prach

SCPI Command :

```
TRIGger:NRMMw:MEASurement<Instance>:PRACh:SOURce
```

class PrachCls

Prach commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NRMMw:MEASurement<Instance>:PRACh:SOURce
value: str = driver.trigger.nrMmw.measurement.prach.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NRMMw:MEASurement<Instance>:PRACh:SOURce
driver.trigger.nrMmw.measurement.prach.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrMmw.measurement.prach.clone()
```

Subgroups

6.16.9.1.2.1 Catalog

SCPI Command :

```
TRIGger:NRMMw:MEASurement<Instance>:PRACh:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NRMMw:MEASurement<Instance>:PRACh:CATalog:SOURce
value: List[str] = driver.trigger.nrMmw.measurement.prach.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.10 NrSub

class NrSubCls

NrSub commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrSub.clone()
```

Subgroups

6.16.10.1 Measurement

class MeasurementCls

Measurement commands group definition. 6 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrSub.measurement.clone()
```

Subgroups

6.16.10.1.1 MultiEval

SCPI Command :

```
TRIGger:NRSub:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.nrSub.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.nrSub.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrSub.measurement.multiEval.clone()
```

Subgroups

6.16.10.1.1.1 Catalog

SCPI Command :

```
TRIGger:NRSub:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.nrSub.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.10.1.2 Prach

SCPI Command :

```
TRIGger:NRSub:MEASurement<Instance>:PRACh:SOURce
```

class PrachCls

Prach commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:PRACh:SOURce
value: str = driver.trigger.nrSub.measurement.prach.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:PRACh:SOURce
driver.trigger.nrSub.measurement.prach.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrSub.measurement.prach.clone()
```

Subgroups

6.16.10.1.2.1 Catalog

SCPI Command :

```
TRIGger:NRSub:MEASurement<Instance>:PRACh:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:PRACh:CATalog:SOURce
value: List[str] = driver.trigger.nrSub.measurement.prach.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.10.1.3 Srs

SCPI Command :

```
TRIGger:NRSub:MEASurement<Instance>:SRS:SOURce
```

class SrsCls

Srs commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:SRS:SOURce
value: str = driver.trigger.nrSub.measurement.srs.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:SRS:SOURce
driver.trigger.nrSub.measurement.srs.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.nrSub.measurement.srs.clone()
```

Subgroups

6.16.10.1.3.1 Catalog

SCPI Command :

```
TRIGger:NRSub:MEASurement<Instance>:SRS:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:NRSub:MEASurement<Instance>:SRS:CATalog:SOURce
value: List[str] = driver.trigger.nrSub.measurement.srs.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.11 Uwb

class UwbCls

Uwb 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.trigger.uwb.clone()
```

Subgroups

6.16.11.1 Measurement

class MeasurementCls

Measurement 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.trigger.uwb.measurement.clone()
```

Subgroups

6.16.11.1.1 MultiEval

SCPI Command :

```
TRIGger:UWB:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:UWB:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.uwb.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:UWB:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.uwb.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.uwb.measurement.multiEval.clone()
```

Subgroups

6.16.11.1.1.1 Catalog

SCPI Command :

```
TRIGger:UWB:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:UWB:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.uwb.measurement.multiEval.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.12 Wcdma

class WcdmaCls

Wcdma commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wcdma.clone()
```

Subgroups

6.16.12.1 Measurement

class MeasurementCls

Measurement commands group definition. 10 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wcdma.measurement.clone()
```

Subgroups

6.16.12.1.1 MultiEval

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.wcdma.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.wcdma.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wcdma.measurement.multiEval.clone()
```

Subgroups

6.16.12.1.1.1 Catalog

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.wcdma.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.12.1.2 OlpControl

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:OLPControl:SOURce
```

class OlpControlCls

OlpControl commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:OLPControl:SOURce
value: str = driver.trigger.wcdma.measurement.olpControl.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:OLPControl:SOURce
driver.trigger.wcdma.measurement.olpControl.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wcdma.measurement.olpControl.clone()
```

Subgroups

6.16.12.1.2.1 Catalog

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:OLPControl:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:OLPControl:CATalog:SOURce
value: List[str] = driver.trigger.wcdma.measurement.olpControl.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.12.1.3 OoSync

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:OOSYnc:SOURce
```

class OoSyncCls

OoSync commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:OOSYnc:SOURce
value: str = driver.trigger.wcdma.measurement.ooSync.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:OOSYnc:SOURce
driver.trigger.wcdma.measurement.ooSync.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wcdma.measurement.ooSync.clone()
```

Subgroups

6.16.12.1.3.1 Catalog

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:OOSYnc:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:OOSYnc:CATalog:SOURce
value: List[str] = driver.trigger.wcdma.measurement.ooSync.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.12.1.4 Prach

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:PRACh:SOURce
```

class PrachCls

Prach commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:PRACh:SOURce
value: str = driver.trigger.wcdma.measurement.prach.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:PRACh:SOURce
driver.trigger.wcdma.measurement.prach.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wcdma.measurement.prach.clone()
```

Subgroups

6.16.12.1.4.1 Catalog

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:PRACh:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:PRACh:CATalog:SOURce
value: List[str] = driver.trigger.wcdma.measurement.prach.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.12.1.5 Tpc

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:TPC:SOURce
```

class TpcCls

Tpc commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:TPC:SOURce
value: str = driver.trigger.wcdma.measurement.tpc.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:TPC:SOURce
driver.trigger.wcdma.measurement.tpc.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wcdma.measurement.tpc.clone()
```

Subgroups

6.16.12.1.5.1 Catalog

SCPI Command :

```
TRIGger:WCDMa:MEASurement<Instance>:TPC:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:WCDMa:MEASurement<Instance>:TPC:CATalog:SOURce
value: List[str] = driver.trigger.wcdma.measurement.tpc.catalog.get_source()
```

No command help available

return
trigger: No help available

6.16.13 Wlan

class WlanCls

Wlan 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.trigger.wlan.clone()
```

Subgroups

6.16.13.1 Measurement

class MeasurementCls

Measurement 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.trigger.wlan.measurement.clone()
```

Subgroups

6.16.13.1.1 MultiEval

SCPI Command :

```
TRIGger:WLAN:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:WLAN:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.wlan.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:WLAN:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.wlan.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wlan.measurement.multiEval.clone()
```

Subgroups

6.16.13.1.1.1 Catalog

SCPI Command :

```
TRIGger:WLAN:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:WLAN:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.wlan.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

6.16.14 Wpan

class WpanCls

Wpan 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.trigger.wpan.clone()
```

Subgroups

6.16.14.1 Measurement

class MeasurementCls

Measurement 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.trigger.wpan.measurement.clone()
```

Subgroups

6.16.14.1.1 MultiEval

SCPI Command :

```
TRIGger:WPAN:MEASurement<Instance>:MEValuation:SOURce
```

class MultiEvalCls

MultiEval commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:WPAN:MEASurement<Instance>:MEValuation:SOURce
value: str = driver.trigger.wpan.measurement.multiEval.get_source()
```

No command help available

return
trigger: No help available

set_source(*trigger: str*) → None

```
# SCPI: TRIGger:WPAN:MEASurement<Instance>:MEValuation:SOURce
driver.trigger.wpan.measurement.multiEval.set_source(trigger = 'abc')
```

No command help available

param trigger
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.wpan.measurement.multiEval.clone()
```

Subgroups

6.16.14.1.1.1 Catalog

SCPI Command :

```
TRIGger:WPAN:MEASurement<Instance>:MEValuation:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:WPAN:MEASurement<Instance>:MEValuation:CATalog:SOURce
value: List[str] = driver.trigger.wpan.measurement.multiEval.catalog.get_
↪source()
```

No command help available

return
trigger: No help available

CHAPTER

SEVEN

RSCMPX_GPRF UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsCMPX_Gprf.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsCMPX_Gprf.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 RsCMPX_Gprf instance with new VISA session, the session gets a new thread lock. You can assign it to other RsCMPX_Gprf sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsCMPX_Gprf 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

RSCMPX_GPRF 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

RSCMPX_GPRF EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

TEN

INDEX

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

R

S

U