RsCmwGprfMeas

Release 4.0.140.33

Rohde & Schwarz

Apr 16, 2024

CONTENTS:


CMW

CHAPTER

ONE

REVISION HISTORY

1.1 RsCmwGprfMeas

Rohde & Schwarz CMW GPRF Measurement RsCmwGprfMeas instrument driver.

Basic Hello-World code:

```
from RsCmwGprfMeas import *

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

Supported instruments: CMW500, CMW270, CMW280, CMW100

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

Documentation: https://RsCmwGprfMeas.readthedocs.io/

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

1.1.1 Version history

Release Notes:

Latest release notes summary: Update for FW 4.0.140

Version 4.0.140

- Update for FW 4.0.140

Version 3.8.xx2

- Fixed several misspelled arguments and command headers

Version 3.8.xx1

- Bluetooth and WLAN update for FW versions 3.8.xxx

Version 3.7.xx8

- Added documentation on ReadTheDocs

Version 3.7.xx7

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

Version 3.7.xx6

- Added new UDF integer number recognition

Version 3.7.xx5

- Added RsCmwDau

Version 3.7.xx4

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

Version 3.7.xx3

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

Version 3.7.xx2

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

Version 1.0.0.0

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsCmwGprfMeas is a Python remote-control communication module for Rohde & Schwarz SCPI-based Test and Measurement Instruments. It represents SCPI commands as fixed APIs and hence provides SCPI autocompletion and helps you to avoid common string typing mistakes.

> Basic example of the idea:
> SCPI command:
> `SYSTem:REFerence:FREQuency:SOURce`
> Python module representation:
> writing:
> `driver.system.reference.frequency.source.set()`
> reading:
> `driver.system.reference.frequency.source.get()`

Check out this RsCmwBase example:

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

from RsCmwBase import *  # install from pypi.org

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

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

(continues on next page)

(continued from previous page)

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

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

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

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


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


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

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

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

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

Couple of reasons why to choose this module over plain SCPI approach:

- Type-safe API using typing module
- You can still use the plain SCPI communication
- You can select which VISA to use or even not use any VISA at all
- Initialization of a new session is straight-forward, no need to set any other properties
- Many useful features are already implemented - reset, self-test, opc-synchronization, error checking, option checking
- Binary data blocks transfer in both directions
- Transfer of arrays of numbers in binary or ASCII format
- File transfers in both directions
- Events generation in case of error, sent data, received data, chunk data (for big files transfer)

- Multithreading session locking - you can use multiple threads talking to one instrument at the same time
- Logging feature tailored for SCPI communication - different for binary and ascii data

2.2 Installation

RsCmwGprfMeas 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 RsCmwGprfMeas`

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 `RsCmwGprfMeas` 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 RsCmwGprfMeas offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwGprfMeas needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwGprfMeas package to your computer from the pypi.org: https://pypi.org/project/RsCmwGprfMeas/#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\RsCmwGprfMeas-4.0.140.33.tar`

2.3 Finding Available Instruments

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

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

from RsCmwGprfMeas import *

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

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

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

- Installed RsCmwGprfMeas Python module Version 4.0.140 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCmwGprfMeas import *

# A good practice is to assure that you have a certain minimum version installed
RsCmwGprfMeas.assert_minimum_version('4.0.140')
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 = RsCmwGprfMeas(resource_string_1)

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

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

from RsCmwGprfMeas import *

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

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

from RsCmwGprfMeas import *

driver = RsCmwGprfMeas('TCPIP::192.168.56.101::5025::SOCKET', True, True, "SelectVisa=
↪'socket'")
print(f'Visa manufacturer: {driver.utilities.visa_manufacturer}')
```

(continues on next page)

(continued from previous page)

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

More `option_string` tokens are separated by comma:

```
driver = RsCmwGprfMeas('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 RsCmwGprfMeas objects:

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

from RsCmwGprfMeas import *

driver1 = RsCmwGprfMeas('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsCmwGprfMeas.from_existing_session(driver1)

print(f'driver1: {driver1.utilities.idn_string}')
print(f'driver2: {driver2.utilities.idn_string}')

# Closing the driver2 session does not close the driver1 session - driver1 is the
↪'session master'
driver2.close()
print(f'driver2: I am closed now')

print(f'driver1: I am  still opened and working: {driver1.utilities.idn_string}')
driver1.close()
print(f'driver1: Only now I am closed.')
```

Note: The `driver1` is the object holding the 'master' session. If you call the `driver1.close()`, the `driver2` loses its instrument session as well, and becomes pretty much useless.

2.5 Plain SCPI Communication

After you have opened the session, you can use the instrument-specific part described in the RsCmwGprfMeas 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 RsCmwGprfMeas import *

driver = RsCmwGprfMeas('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 RsCmwGprfMeas import *

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

(continues on next page)

(continued from previous page)

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

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

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

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

driver = RsCmwGprfMeas('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

RsCmwGprfMeas 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 RsCmwGprfMeas 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 RsCmwGprfMeas import *

```

(continues on next page)

(continued from previous page)

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

RsCmwGprfMeas 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 RsCmwGprfMeas 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 = RsCmwGprfMeas('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 RsCmwGprfMeas 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, RsCmwGprfMeas 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 RsCmwGprfMeas object
"""

import threading
from RsCmwGprfMeas import *
```

(continues on next page)

(continued from previous page)

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


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

import threading
from RsCmwGprfMeas import *


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

Run the following example:

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

import threading
from RsCmwGprfMeas import *


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

driver = RsCmwGprfMeas('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 = RsCmwGprfMeas('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 RsCmwGprfMeas import *

driver = RsCmwGprfMeas('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 RsCmwGprfMeas import *

driver = RsCmwGprfMeas('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 AveragingMode

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

3.2 CcdfMode

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

3.3 CmwsConnector

```
# First value:
value = enums.CmwsConnector.R11
# Last value:
value = enums.CmwsConnector.RH8
# All values (96x):
R11 | R12 | R13 | R14 | R15 | R16 | R17 | R18
R21 | R22 | R23 | R24 | R25 | R26 | R27 | R28
R31 | R32 | R33 | R34 | R35 | R36 | R37 | R38
R41 | R42 | R43 | R44 | 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 | RF2 | RF3 | RF4 | RF5 | RF6 | RF7 | RF8
RG1 | RG2 | RG3 | RG4 | RG5 | RG6 | RG7 | RG8
RH1 | RH2 | RH3 | RH4 | RH5 | RH6 | RH7 | RH8
```

3.4 Detector

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

3.5 DetectorBasic

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

3.6 DigitalFilterType

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

3.7 FileSave

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

3.8 FilterType

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

3.9 IqFormat

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

3.10 MagnitudeUnit

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

3.11 MeasTab

```
# Example value:
value = enums.MeasTab.EPSensor
# All values (6x):
EPSensor | FFTSanalyzer | IQRecorder | IQVSlot | POWer | SPECtrum
```

3.12 MeasurementMode

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

3.13 OffsetMode

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

3.14 ParameterSetMode

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

3.15 PathIndex

```
# Example value:
value = enums.PathIndex.P1
# All values (8x):
P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8
```

3.16 PathLossState

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

3.17 PwrSensorResolution

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

3.18 RbwFilterType

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

3.19 Repeat

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

3.20 ResourceState

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

3.21 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
```

3.22 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
```

(continues on next page)

(continued from previous page)

```
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
```

3.23 RxConverter

```
# First value:
value = enums.RxConverter.IRX1
# Last value:
value = enums.RxConverter.RX44
# All values (40x):
IRX1 | IRX11 | IRX12 | IRX13 | IRX14 | IRX2 | IRX21 | IRX22
IRX23 | IRX24 | IRX3 | IRX31 | IRX32 | IRX33 | IRX34 | IRX4
IRX41 | IRX42 | IRX43 | IRX44 | RX1 | RX11 | RX12 | RX13
RX14 | RX2 | RX21 | RX22 | RX23 | RX24 | RX3 | RX31
RX32 | RX33 | RX34 | RX4 | RX41 | RX42 | RX43 | RX44
```

3.24 Scenario

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

3.25 SignalSlopeExt

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

3.26 SpanMode

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

3.27 Statistic

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

3.28 TargetMainState

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

3.29 TargetSyncState

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

3.30 Timing

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

3.31 TriggerPowerMode

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

3.32 TriggerSequenceMode

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

3.33 TriggerSource

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

3.34 UserDebugMode

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

3.35 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 Marker

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

4.3 Sensor

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

CHAPTER

FIVE

EXAMPLES

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

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

from RsCmwBase import *  # install from pypi.org

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

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

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

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

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

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


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

(continues on next page)

(continued from previous page)

```
↪{event_args.message}')


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

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

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

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

CHAPTER

SIX

RSCMWGPRFMEAS API STRUCTURE

Global RepCaps

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

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

410 total commands, 14 Subgroups, 0 group commands

Initializes new RsCmwGprfMeas 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 RsCmwGprfMeas.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 RsCmwGprfMeas session.

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

Creates a new RsCmwGprfMeas 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 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

6.1.1 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.extPwrSensor.clone()
```

Subgroups

6.1.1.1 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.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) .

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

return
zeroing_state: PASSed | FAILed '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.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) .

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

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

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
driver.calibration.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) .

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

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

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

6.2 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.canalyzer.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>:CANalyzer
driver.canalyzer.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>:CANalyzer
driver.canalyzer.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.canalyzer.clone()
```

Subgroups

6.2.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:CANalyzer:STATe
value: enums.ResourceState = driver.canalyzer.state.fetch(timeout = 1.0, target_
↪main_state = enums.TargetMainState.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.canalyzer.state.clone()
```

Subgroups

6.2.1.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:CANalyzer:STATe:ALL
value: List[enums.ResourceState] = driver.canalyzer.state.all.fetch(timeout = 1.
↪0, target_main_state = enums.TargetMainState.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

6.3 Configure

SCPI Command :

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

class ConfigureCls

Configure commands group definition. 152 total commands, 10 Subgroups, 1 group commands

get_display() → MeasTab

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:DISPlay
value: enums.MeasTab = driver.configure.get_display()
```

Selects the displayed measurement tab. This command is useful, if you want to observe the GUI during remote control. The GUI controls are disabled in that case, so that you cannot select a tab via the GUI. To display the GUI, use SYSTem:DISPlay:UPDate ON.

return
meas_tab: POWer | SPECtrum | FFTSanalyzer | IQRecorder | IQVSlot | EPSensor

set_display(*meas_tab: MeasTab*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:DISPlay
driver.configure.set_display(meas_tab = enums.MeasTab.EPSensor)
```

Selects the displayed measurement tab. This command is useful, if you want to observe the GUI during remote control. The GUI controls are disabled in that case, so that you cannot select a tab via the GUI. To display the GUI, use SYSTem:DISPlay:UPDate ON.

param meas_tab
POWer | SPECtrum | FFTSanalyzer | IQRecorder | IQVSlot | EPSensor

Cloning the Group

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

Subgroups

6.3.1 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.canalyzer.get_mname()
```

No command help available

return
meas_name: No help available

get_segment() → int

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

No command help available

return
segment: No help available

get_step() → int

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

No command help available

return
step: No help available

set_segment(*segment: int*) → None

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

No command help available

param segment
No help available

set_step(*step: int*) → None

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

No command help available

param step
No help available

Cloning the Group

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

Subgroups

6.3.1.1 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.canalyzer.iqFile.get()
```

No command help available

return
filename_return: No help available

set(*filename: str*) → None

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

No command help available

param filename
No help available

6.3.1.2 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.canalyzer.sall.get_iq_folder()
```

No command help available

return
folder_name: No help available

get_wt_folder() → bool

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

No command help available

return
write_to_folder: No help available

set_iq_folder(*folder_name: str*) → None

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

No command help available

param folder_name
No help available

set_wt_folder(*write_to_folder: bool*) → None

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

No command help available

param write_to_folder
No help available

6.3.2 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. 7 total commands, 1 Subgroups, 5 group commands

get_frequency() → float

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

Specifies the input frequency at the power sensor.

return
correction_freq: numeric Range: Depends on sensor model , Unit: Hz

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:REPetition
value: enums.Repeat = driver.configure.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_resolution() → PwrSensorResolution

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RESolution
value: enums.PwrSensorResolution = driver.configure.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 | PD1 | PD2 | PD3 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.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: numeric Number of measurement intervals Range: 1 to 1000

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:TOUT
value: float = driver.configure.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: numeric Unit: s

set_frequency(*correction_freq: float*) → None

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

Specifies the input frequency at the power sensor.

param correction_freq
numeric Range: Depends on sensor model , Unit: Hz

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:REPetition
driver.configure.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_resolution(*resolution: PwrSensorResolution*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RESolution
driver.configure.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 | PD1 | PD2 | PD3 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.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
numeric Number of measurement intervals Range: 1 to 1000

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:TOUT
driver.configure.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
numeric Unit: s

Cloning the Group

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

Subgroups

6.3.2.1 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.extPwrSensor.attenuation.get_state()
```

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

return
attenuator_state: OFF | ON

get_value() → float

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

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

return
attenuation: numeric Range: -50 dB to 50 dB, Unit: dB

set_state(*attenuator_state: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation:STATe
driver.configure.extPwrSensor.attenuation.set_state(attenuator_state = False)
```

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

param attenuator_state
OFF | ON

set_value(*attenuation: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation
driver.configure.extPwrSensor.attenuation.set_value(attenuation = 1.0)
```

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

param attenuation
numeric Range: -50 dB to 50 dB, Unit: dB

6.3.3 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.fftSpecAn.get_amode()
```

Selects the averaging mode for the average spectrum trace.

return
averaging_mode: LINear | LOGarithmic 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.fftSpecAn.get_detector()
```

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

return
detector: PEAK | RMS 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.fftSpecAn.get_fft_length()
```

Selects the number of samples recorded per measurement interval.

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

get_fspan() → float

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

Configures the frequency span of the FFT spectrum analyzer.

return
frequency_span: numeric Only the following values can be configured, all values in MHz: 1.25, 2.5, 5, 10, 20, 40, 80, 160 Other values are rounded to the next allowed value. Unit: Hz

get_mo_exception() → bool

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

Specifies whether measurement results that the R&S CMW identifies as faulty or inaccurate are rejected.

return
meas_on_exception: OFF | ON 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.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
value: int = driver.configure.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: numeric Number of measurement intervals Range: 1 to 1000

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
value: float = driver.configure.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: numeric Unit: s

set_amode(*averaging_mode: AveragingMode*) → None

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

Selects the averaging mode for the average spectrum trace.

param averaging_mode
LINear | LOGarithmic 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.fftSpecAn.set_detector(detector = enums.DetectorBasic.PEAK)
```

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

param detector
PEAK | RMS 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.fftSpecAn.set_fft_length(length = 1)
```

Selects the number of samples recorded per measurement interval.

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

set_fspan(*frequency_span: float*) → None

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

Configures the frequency span of the FFT spectrum analyzer.

param frequency_span
numeric Only the following values can be configured, all values in MHz: 1.25, 2.5, 5, 10, 20, 40, 80, 160 Other values are rounded to the next allowed value. Unit: Hz

set_mo_exception(*meas_on_exception: bool*) → None

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

Specifies whether measurement results that the R&S CMW identifies as faulty or inaccurate are rejected.

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

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:REPetition
driver.configure.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
driver.configure.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
numeric Number of measurement intervals Range: 1 to 1000

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
driver.configure.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
numeric Unit: s

Cloning the Group

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

Subgroups

6.3.3.1 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: OFF | ON 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: numeric Lower end of the search range for marker 0. Unit: Hz
- Peak_Range_To_0: float: numeric Upper end of the search range for marker 0. Unit: Hz
- Full_Span_Enable_1: bool: OFF | ON Enable full-span search for marker 1.
- Peak_Range_From_1: float: numeric Lower end of the search range for marker 1. Unit: Hz
- Peak_Range_To_1: float: numeric Upper end of the search range for marker 1. Unit: Hz
- Full_Span_Enable_2: bool: OFF | ON Enable full-span search for marker 2.
- Peak_Range_From_2: float: numeric Lower end of the search range for marker 2. Unit: Hz
- Peak_Range_To_2: float: numeric Upper end of the search range for marker 2. Unit: Hz
- Full_Span_Enable_3: bool: OFF | ON Enable full-span search for marker 3. Unit: Hz
- Peak_Range_From_3: float: numeric Lower end of the search range for marker 3. Unit: Hz
- Peak_Range_To_3: float: numeric Upper end of the search range for marker 3. Unit: Hz
- Full_Span_Enable_4: bool: OFF | ON Enable full-span search for marker 4.
- Peak_Range_From_4: float: numeric Lower end of the search range for marker 4. Unit: Hz
- Peak_Range_To_4: float: numeric Upper end of the search range for marker 4. Unit: Hz

get_noa_markers() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch:NOAMarkers
value: int = driver.configure.fftSpecAn.peakSearch.get_noa_markers()
```

Defines the number of active markers for the peak search.

return
no_active_markers: numeric Range: 0 to 5

get_value() → ValueStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch
value: ValueStruct = driver.configure.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.fftSpecAn.peakSearch.set_noa_markers(no_active_markers = 1)
```

Defines the number of active markers for the peak search.

param no_active_markers
numeric Range: 0 to 5

set_value(*value: ValueStruct*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch
structure = driver.configure.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.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.

6.3.4 IqRecorder

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MODE
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:RATio
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
```

class IqRecorderCls

IqRecorder commands group definition. 23 total commands, 4 Subgroups, 8 group commands

get_format_py() → IqFormat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FORMat
value: enums.IqFormat = driver.configure.iqRecorder.get_format_py()
```

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

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

get_iq_file() → str

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

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

return
iq_save_file: string 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.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.iqRecorder.get_munit()
```

Selects the magnitude unit for the measurement results.

return
magnitude_unit: VOLT | RAW 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.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: numeric

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
value: float = driver.configure.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: numeric Unit: s

get_user() → UserDebugMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:USER
value: enums.UserDebugMode = driver.configure.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.iqRecorder.get_wt_file()
```

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

return
write_to_iq_file: OFF | ON | ONLY 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_format_py(*format_py: IqFormat*) → None

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

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

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

set_iq_file(*iq_save_file: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQFile
driver.configure.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
string 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.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.iqRecorder.set_munit(magnitude_unit = enums.MagnitudeUnit.RAW)
```

Selects the magnitude unit for the measurement results.

param magnitude_unit
VOLT | RAW 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.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
numeric

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
driver.configure.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
numeric Unit: s

set_user(*user_mode: UserDebugMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:USER
driver.configure.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.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 RsCmwGprfMeas. Configure.IqRecorder.iqFile.

param write_to_iq_file
OFF | ON | ONLY 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.iqRecorder.clone()
```

Subgroups

6.3.4.1 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: integer Samples before trigger event Range: 1 to 4194303
- Capt_Samp_Aft_Trig: int: integer Samples after trigger event Range: 1 to 4194303

get() → CaptureStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:CAPTure
value: CaptureStruct = driver.configure.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.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
integer Samples before trigger event Range: 1 to 4194303

param capt_samp_aft_trig
integer Samples after trigger event Range: 1 to 4194303

6.3.4.2 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.iqRecorder.filterPy.get_type_py()
```

Selects the IF filter type.

return
filter_type: BANDpass | GAUSs 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.iqRecorder.filterPy.set_type_py(filter_type = enums.
↪RbwFilterType.BANDpass)
```

Selects the IF filter type.

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

Cloning the Group

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

Subgroups

6.3.4.2.1 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.iqRecorder.filterPy.bandpass.get_bandwidth()
```

Selects the bandwidth for a bandpass filter.

return
bandpass_bw: numeric Only the following values can be configured: 1, 10, 100 kHz; 1, 10, 40, 160 MHz Other values are rounded to the next allowed value. Unit: Hz

set_bandwidth(*bandpass_bw: float*) → None

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

Selects the bandwidth for a bandpass filter.

param bandpass_bw
numeric Only the following values can be configured: 1, 10, 100 kHz; 1, 10, 40, 160 MHz Other values are rounded to the next allowed value. Unit: Hz

6.3.4.2.2 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.iqRecorder.filterPy.gauss.get_bandwidth()
```

Selects the bandwidth for a filter of Gaussian shape.

return
gauss_bw: numeric 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. Unit: Hz

set_bandwidth(*gauss_bw: float*) → None

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

Selects the bandwidth for a filter of Gaussian shape.

param gauss_bw
numeric 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. Unit: Hz

6.3.4.3 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.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.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.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.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.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.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.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.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.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.iqRecorder.listPy.clone()
```

Subgroups

6.3.4.3.1 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.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.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.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.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

6.3.4.3.2 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.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.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.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.iqRecorder.listPy.frequency.set_all(frequency = [1.1, 2.2, 3.
↪3])
```

No command help available

param frequency
No help available

6.3.4.3.3 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.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.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

6.3.4.4 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.iqRecorder.trigger.get_source()
```

No command help available

return
source: No help available

set_source(*source: TriggerSource*) → None

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

No command help available

param source
No help available

6.3.5 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.iqVsSlot.get_fe_limit()
```

Defines the frequency estimation limit as 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: numeric Range: -100 dB to 0 dB, Unit: dB

get_ftype() → FilterType

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

Selects the IF filter type.

return
filter_type: GAUSs | NYQuist | NY1Mhz 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.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: numeric Range: 10E-6 s to StepLength, Unit: s

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:REPetition
value: enums.Repeat = driver.configure.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SCOunt
value: int = driver.configure.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: integer Range: 1 to 3000

get_slength() → float

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

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

return
step_length: numeric Range: MeasLength to 5E-3 s, Unit: s

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:TOUT
value: float = driver.configure.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: numeric Unit: s

set_fe_limit(*limit: float*) → None

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

Defines the frequency estimation limit as 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
numeric Range: -100 dB to 0 dB, Unit: dB

set_ftype(*filter_type: FilterType*) → None

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

Selects the IF filter type.

param filter_type
GAUSs | NYQuist | NY1Mhz 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.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
numeric Range: 10E-6 s to StepLength, Unit: s

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:REPetition
driver.configure.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*step_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:SCOunt
driver.configure.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
integer Range: 1 to 3000

set_slength(*step_length: float*) → None

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

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

param step_length
numeric Range: MeasLength to 5E-3 s, Unit: s

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:TOUT
driver.configure.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
numeric Unit: s

Cloning the Group

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

Subgroups

6.3.5.1 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.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: decimal Range: 1 to 200

get_start() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STARt
value: int = driver.configure.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: numeric Range: 0 to StopIndex

get_stop() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STOP
value: int = driver.configure.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: numeric Range: StartIndex to 199

get_value() → bool

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

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

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

set_start(*start_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STARt
driver.configure.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
numeric Range: 0 to StopIndex

set_stop(*stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:STOP
driver.configure.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
numeric Range: StartIndex to 199

set_value(*list_mode: bool*) → None

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

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

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

Cloning the Group

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

Subgroups

6.3.5.1.1 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.iqVsSlot.listPy.envelopePower.get(index = 1)
```

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

param index
integer Range: 0 to 199

return
exp_nom_power: numeric 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 data sheet. Unit: dBm

get_all() → List[float]

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

Defines the expected nominal power for all subsweeps.

return
exp_nom_power: numeric 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 data sheet. Unit: dBm

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower
driver.configure.iqVsSlot.listPy.envelopePower.set(index = 1, exp_nom_power = 1.
↪0)
```

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

param index
integer Range: 0 to 199

param exp_nom_power
numeric 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 data sheet. Unit: dBm

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:ENPower:ALL
driver.configure.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
numeric 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 data sheet. Unit: dBm

6.3.5.1.2 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.iqVsSlot.listPy.frequency.get(index = 1)
```

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

param index
integer Range: 0 to 199

return
frequency: numeric Unit: Hz

get_all() → List[float]

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

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

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

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency
driver.configure.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
integer Range: 0 to 199

param frequency
numeric Unit: Hz

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:FREQuency:ALL
driver.configure.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
numeric Comma-separated list of frequencies, one value per subsweep Unit: Hz

6.3.5.1.3 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.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
integer Range: 0 to 199

return
retrigger: OFF | ON

get_all() → List[bool]

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

Configures the retrigger mechanism for all subsweeps. The setting is only relevant for trigger mode 'Retrigger Preselect'.

return
retrigger: OFF | ON 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.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
integer Range: 0 to 199

param retrigger
OFF | ON

set_all(*retrigger: List[bool]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:RETRigger:ALL
driver.configure.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
OFF | ON Comma-separated list of values, one value per subsweep

6.3.5.1.4 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: numeric Range: 0 to StopIndex
- Stop_Index: int: numeric Range: StartIndex to 199

get() → SstopStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:LIST:SSTop
value: SstopStruct = driver.configure.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.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
numeric Range: 0 to StopIndex

param stop_index
numeric Range: StartIndex to 199

6.3.5.2 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.iqVsSlot.trigger.get_source()
```

No command help available

return
source: No help available

set_source(*source: TriggerSource*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQVSlot:TRIGger:SOURce
driver.configure.iqVsSlot.trigger.set_source(source = enums.TriggerSource.
↪EXTernal)
```

No command help available

param source
No help available

6.3.6 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.nrpm.get_repetition()
```

No command help available

return
repetition: No help available

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:SCOunt
value: int = driver.configure.nrpm.get_scount()
```

No command help available

return
statistic_count: No help available

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:TOUT
value: float = driver.configure.nrpm.get_timeout()
```

No command help available

return
tcd_timeout: No help available

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRPM:REPetition
driver.configure.nrpm.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>:NRPM:SCOunt
driver.configure.nrpm.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>:NRPM:TOUT
driver.configure.nrpm.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.nrpm.clone()
```

Subgroups

6.3.6.1 Sensor<Sensor>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.nrpm.sensor.repcap_sensor_get()
driver.configure.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.nrpm.sensor.clone()
```

Subgroups

6.3.6.1.1 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.nrpm.sensor.frequency.get(sensor = repcap.
↪Sensor.Default)
```

No command help available

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.nrpm.sensor.frequency.set(frequency = 1.0, sensor = repcap.
↪Sensor.Default)
```

No command help available

param frequency
No help available

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

6.3.7 Ploss

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:PLOSs:TRACe
```

class PlossCls

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

get_trace() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TRACe
value: bool = driver.configure.ploss.get_trace()
```

No command help available

return
trace_mode: No help available

set_trace(*trace_mode: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:TRACe
driver.configure.ploss.set_trace(trace_mode = False)
```

No command help available

param trace_mode
No help available

Cloning the Group

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

Subgroups

6.3.7.1 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.ploss.listPy.clone()
```

Subgroups

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

get(*connector: CmwsConnector*) → GetStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:LIST:FREQuency
value: GetStruct = driver.configure.ploss.listPy.frequency.get(connector =␣
↪enums.CmwsConnector.R11)
```

No command help available

param connector
No help available

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

set(*connector: CmwsConnector*, *num_entries: int*, *frequency: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:LIST:FREQuency
driver.configure.ploss.listPy.frequency.set(connector = enums.CmwsConnector.R11,
↪ num_entries = 1, frequency = [1.1, 2.2, 3.3])
```

No command help available

param connector
No help available

param num_entries
No help available

param frequency
No help available

6.3.7.2 Mpath

class MpathCls

Mpath 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.ploss.mpath.clone()
```

Subgroups

6.3.7.2.1 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.ploss.mpath.listPy.clone()
```

Subgroups

6.3.7.2.1.1 Frequency

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:PLOSs:MPATh:LIST:FREQuency
```

class FrequencyCls

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

class GetStruct

Response structure. Fields:

- Num_Entries: int: No parameter help available
- Frequency: List[float]: No parameter help available

get(*connector: CmwsConnector*, *path_index: PathIndex*) → GetStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:MPATh:LIST:FREQuency
value: GetStruct = driver.configure.ploss.mpath.listPy.frequency.get(connector
↪= enums.CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

param connector
No help available

param path_index
No help available

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

set(*connector: CmwsConnector*, *path_index: PathIndex*, *num_entries: int*, *frequency: List[float]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:MPATh:LIST:FREQuency
driver.configure.ploss.mpath.listPy.frequency.set(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1, num_entries = 1,
↪frequency = [1.1, 2.2, 3.3])
```

No command help available

param connector
No help available

param path_index
No help available

param num_entries
No help available

param frequency
No help available

6.3.7.3 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.ploss.view.get_aftaps()
```

No command help available

return
avg_filter_taps: No help available

set_aftaps(*avg_filter_taps: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:PLOSs:VIEW:AFTaps
driver.configure.ploss.view.set_aftaps(avg_filter_taps = 1)
```

No command help available

param avg_filter_taps
No help available

6.3.8 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. 48 total commands, 5 Subgroups, 7 group commands

get_mlength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MLENgth
value: float = driver.configure.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: numeric Unit: s

get_mode() → CcdfMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MODE
value: enums.CcdfMode = driver.configure.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.power.get_pdef_set()
```

This command is related to the global parameter set. A setting command loads a predefined set of parameters into the global parameter set. A query returns the name of the predefined set assigned to the global parameter set. To get a list of predefined-set strings, use method RsCmwGprfMeas.Configure.Power.ParameterSetList.Catalog.pdefSet.

return
predefined_set: string

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:REPetition
value: enums.Repeat = driver.configure.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
value: int = driver.configure.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: numeric Number of measurement intervals Range: 1 to 100E+3

get_slength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SLENgth
value: float = driver.configure.power.get_slength()
```

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

return
step_length: numeric Unit: s

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:TOUT
value: float = driver.configure.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: numeric Unit: s

set_mlength(*meas_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MLENgth
driver.configure.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
numeric Unit: s

set_mode(*ccdf_mode: CcdfMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MODE
driver.configure.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 | STATistic POWer: 'Power' mode STATistic: 'Statistic' mode

set_pdef_set(*predefined_set: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PDEFset
driver.configure.power.set_pdef_set(predefined_set = 'abc')
```

This command is related to the global parameter set. A setting command loads a predefined set of parameters into the global parameter set. A query returns the name of the predefined set assigned to the global parameter set. To get a list of predefined-set strings, use method RsCmwGprfMeas.Configure.Power.ParameterSetList.Catalog.pdefSet.

param predefined_set
string

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:REPetition
driver.configure.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
driver.configure.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
numeric Number of measurement intervals Range: 1 to 100E+3

set_slength(*step_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SLENgth
driver.configure.power.set_slength(step_length = 1.0)
```

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

param step_length
numeric Unit: s

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:TOUT
driver.configure.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
numeric Unit: s

Cloning the Group

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

Subgroups

6.3.8.1 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.power.catalog.get_pdef_set()
```

Gets a comma/separated list of predefined parameter sets that can be loaded using method RsCmwGprfMeas.Configure.Power. pdefSet. See also 'Predefined parameter sets'.

return
predefined_set: string Comma-separated list of strings

6.3.8.2 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.power.filterPy.get_type_py()
```

Selects the IF filter type.

return
filter_type: 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.power.filterPy.set_type_py(filter_type = enums.
↪DigitalFilterType.BANDpass)
```

Selects the IF filter type.

param filter_type
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.power.filterPy.clone()
```

Subgroups

6.3.8.2.1 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.power.filterPy.bandpass.get_bandwidth()
```

Selects the bandwidth for a bandpass filter.

return
bandpass_bw: numeric For allowed values, see Table 'Supported values'. Unit: Hz

set_bandwidth(*bandpass_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:BANDpass:BWIDth
driver.configure.power.filterPy.bandpass.set_bandwidth(bandpass_bw = 1.0)
```

Selects the bandwidth for a bandpass filter.

param bandpass_bw
numeric For allowed values, see Table 'Supported values'. Unit: Hz

6.3.8.2.2 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.power.filterPy.gauss.get_bandwidth()
```

Selects the bandwidth for a filter of Gaussian shape.

return
gauss_bw: numeric For allowed values, see Table 'Supported values'. Unit: Hz

set_bandwidth(*gauss_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:GAUSs:BWIDth
driver.configure.power.filterPy.gauss.set_bandwidth(gauss_bw = 1.0)
```

Selects the bandwidth for a filter of Gaussian shape.

param gauss_bw
numeric For allowed values, see Table 'Supported values'. Unit: Hz

6.3.8.3 ListPy

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXITiming
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
```

class ListPyCls

ListPy commands group definition. 23 total commands, 8 Subgroups, 6 group commands

get_count() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:COUNt
value: int = driver.configure.power.listPy.get_count()
```

Queries the total number of segments per sweep, including repetitions.

return
result_count: decimal Range: 1 to 10000

get_munit() → MagnitudeUnit

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:MUNit
value: enums.MagnitudeUnit = driver.configure.power.listPy.get_munit()
```

No command help available

return
magnitude_unit: No help available

get_start() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STARt
value: int = driver.configure.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: numeric Range: 0 to StopIndex

get_stop() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STOP
value: int = driver.configure.power.listPy.get_stop()
```

Selects the last segment to be measured (end of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

return
stop_index: numeric Range: StartIndex to 3999

get_txi_timing() → Timing

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXITiming
value: enums.Timing = driver.configure.power.listPy.get_txi_timing()
```

Specifies the timing of the generated 'GPRF Meas<i>:Power' trigger.

return
timing: STEP | CENTered 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.power.listPy.get_value()
```

Enables or disables the list mode for the power measurement.

return
enable_list_mode: OFF | ON OFF: list mode off ON: list mode on

set_munit(*magnitude_unit: MagnitudeUnit*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:MUNit
driver.configure.power.listPy.set_munit(magnitude_unit = enums.MagnitudeUnit.
↪RAW)
```

No command help available

param magnitude_unit
No help available

set_start(*start_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STARt
driver.configure.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
numeric Range: 0 to StopIndex

set_stop(*stop_index: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:STOP
driver.configure.power.listPy.set_stop(stop_index = 1)
```

Selects the last segment to be measured (end of a sweep) . The total number of segments per sweep, including repetitions, must not be higher than 10000.

param stop_index
numeric Range: StartIndex to 3999

set_txi_timing(*timing: Timing*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:TXITiming
driver.configure.power.listPy.set_txi_timing(timing = enums.Timing.CENTered)
```

Specifies the timing of the generated 'GPRF Meas<i>:Power' trigger.

param timing
STEP | CENTered 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.power.listPy.set_value(enable_list_mode = False)
```

Enables or disables the list mode for the power measurement.

param enable_list_mode
OFF | ON OFF: list mode off ON: list mode on

Cloning the Group

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

Subgroups

6.3.8.3.1 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.power.listPy.envelopePower.get(index = 1)
```

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

param index
integer Range: 0 to 3999

return
exp_nom_power: numeric 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 data sheet. Unit: dBm

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower:ALL
value: List[float] = driver.configure.power.listPy.envelopePower.get_all()
```

Defines the expected nominal power for all segments.

return
exp_nom_power: numeric 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 data sheet. Unit: dBm

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower
driver.configure.power.listPy.envelopePower.set(index = 1, exp_nom_power = 1.0)
```

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

param index
integer Range: 0 to 3999

param exp_nom_power
numeric 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 data sheet. Unit: dBm

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:ENPower:ALL
driver.configure.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
numeric 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 data sheet. Unit: dBm

6.3.8.3.2 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.power.listPy.frequency.get(index = 1)
```

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

param index
integer Range: 0 to 3999

return
frequency: numeric Unit: Hz

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency:ALL
value: List[float] = driver.configure.power.listPy.frequency.get_all()
```

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

return
frequency: numeric Comma-separated list of frequencies, one value per segment Unit: Hz

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency
driver.configure.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
integer Range: 0 to 3999

param frequency
numeric Unit: Hz

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:FREQuency:ALL
driver.configure.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
numeric Comma-separated list of frequencies, one value per segment Unit: Hz

6.3.8.3.3 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.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.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.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.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.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.power.listPy.iqData.set_capture(capture_iq_data = False)
```

No command help available

param capture_iq_data
No help available

6.3.8.3.4 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.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
integer Range: 0 to 3999

return
repetition: numeric Range: 1 to 10000

`get_all`() → List[int]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition:ALL
value: List[int] = driver.configure.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: numeric Comma-separated list of repetitions, one value per segment Range: 1 to 10000

`set`(*index: int*, *repetition: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition
driver.configure.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
integer Range: 0 to 3999

param repetition
numeric Range: 1 to 10000

set_all(*repetition: List[int]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:IREPetition:ALL
driver.configure.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
numeric Comma-separated list of repetitions, one value per segment Range: 1 to 10000

6.3.8.3.5 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.power.listPy.parameterSetList.get(index = 1)
```

Selects the parameter set for segment <Index>.

param index
integer Range: 0 to 3999

return
parameter_set: numeric Range: 0 to 31

get_all() → List[int]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET:ALL
value: List[int] = driver.configure.power.listPy.parameterSetList.get_all()
```

Selects the parameter set for all segments.

return
parameter_set: numeric Comma-separated list of parameter set numbers, one value per segment. Range: 0 to 31

set(*index: int*, *parameter_set: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET
driver.configure.power.listPy.parameterSetList.set(index = 1, parameter_set = 1)
```

Selects the parameter set for segment <Index>.

param index
integer Range: 0 to 3999

param parameter_set
numeric Range: 0 to 31

set_all(*parameter_set: List[int]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:PSET:ALL
driver.configure.power.listPy.parameterSetList.set_all(parameter_set = [1, 2,␣
↪3])
```

Selects the parameter set for all segments.

param parameter_set
numeric Comma-separated list of parameter set numbers, one value per segment. Range: 0 to 31

6.3.8.3.6 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.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
integer Range: 0 to 3999

return
retrigger: OFF | ON

get_all() → List[bool]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger:ALL
value: List[bool] = driver.configure.power.listPy.retrigger.get_all()
```

Configures the retrigger mechanism for all segments. The setting is only relevant for trigger mode 'Retrigger Preselect'.

return
retrigger: OFF | ON 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.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
integer Range: 0 to 3999

param retrigger
OFF | ON

set_all(*retrigger: List[bool]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:RETRigger:ALL
driver.configure.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
OFF | ON Comma-separated list of values, one value per segment

6.3.8.3.7 SingleCmw

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CMODe
```

class SingleCmwCls

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

get_cmode() → ParameterSetMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CMODe
value: enums.ParameterSetMode = driver.configure.power.listPy.singleCmw.get_
↪cmode()
```

No command help available

return
cmws_connector_mode: No help available

set_cmode(*cmws_connector_mode: ParameterSetMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CMODe
driver.configure.power.listPy.singleCmw.set_cmode(cmws_connector_mode = enums.
↪ParameterSetMode.GLOBal)
```

No command help available

param cmws_connector_mode
No help available

Cloning the Group

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

Subgroups

6.3.8.3.7.1 Connector

SCPI Commands :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CONNector
CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CONNector:ALL
```

class ConnectorCls

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

get(*index: int*) → CmwsConnector

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CONNector
value: enums.CmwsConnector = driver.configure.power.listPy.singleCmw.connector.
↪get(index = 1)
```

No command help available

param index
No help available

return
cmws_connector: No help available

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CONNector:ALL
value: List[float] = driver.configure.power.listPy.singleCmw.connector.get_all()
```

No command help available

return
cmws_connector: No help available

set(*index: int*, *cmws_connector: CmwsConnector*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CONNector
driver.configure.power.listPy.singleCmw.connector.set(index = 1, cmws_connector␣
↪= enums.CmwsConnector.R11)
```

No command help available

param index
No help available

param cmws_connector
No help available

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:CMWS:CONNector:ALL
driver.configure.power.listPy.singleCmw.connector.set_all(cmws_connector = [1.1,
↪ 2.2, 3.3])
```

No command help available

param cmws_connector
No help available

6.3.8.3.8 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: numeric Range: 0 to StopIndex
- Stop_Index: int: numeric Range: StartIndex to 3999

get() → SstopStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:LIST:SSTop
value: SstopStruct = driver.configure.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.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
numeric Range: 0 to StopIndex

param stop_index
numeric Range: StartIndex to 3999

6.3.8.4 ParameterSetList

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:POWer:PSET
```

class ParameterSetListCls

ParameterSetList commands group definition. 13 total commands, 5 Subgroups, 1 group commands

get_value() → ParameterSetMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET
value: enums.ParameterSetMode = driver.configure.power.parameterSetList.get_
↪value()
```

Selects whether all segments use the same measurement control settings.

return
parameter_set_mode: GLOBal | LIST 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.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 | LIST 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.power.parameterSetList.clone()
```

Subgroups

6.3.8.4.1 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.power.parameterSetList.catalog.get_pdef_
↪set()
```

Gets a comma-separated list of predefined parameter sets that can be loaded using method RsCmwGprfMeas.Configure.Power.ParameterSetList.PdefSet.set. See also 'Predefined parameter sets'.

return
predefined_set: string Comma-separated list of strings

6.3.8.4.2 FilterPy

class FilterPyCls

FilterPy 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.configure.power.parameterSetList.filterPy.clone()
```

Subgroups

6.3.8.4.2.1 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.power.parameterSetList.filterPy.bandpass.clone()
```

Subgroups

6.3.8.4.2.2 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.power.parameterSetList.filterPy.bandpass.
↪bandwidth.get(index = 1)
```

Selects the bandpass filter bandwidth for the parameter set <Index>.

param index
integer Range: 0 to 31

return
bandwidth: numeric For supported values, see Table 'Supported values'. Unit: Hz

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>
↪:POWer:PSET:FILTer:BANDpass:BWIDth:ALL
value: List[float] = driver.configure.power.parameterSetList.filterPy.bandpass.
↪bandwidth.get_all()
```

Selects the bandpass filter bandwidth for all parameter sets.

return
bandwidth: numeric Comma-separated list of 32 values, for parameter set 0 to 31 For supported values, see Table 'Supported values'. Unit: Hz

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:BANDpass:BWIDth
driver.configure.power.parameterSetList.filterPy.bandpass.bandwidth.set(index =␣
↪1, bandwidth = 1.0)
```

Selects the bandpass filter bandwidth for the parameter set <Index>.

param index
integer Range: 0 to 31

param bandwidth
numeric For supported values, see Table 'Supported values'. Unit: Hz

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>
↪:POWer:PSET:FILTer:BANDpass:BWIDth:ALL
driver.configure.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
numeric Comma-separated list of 32 values, for parameter set 0 to 31 For supported values, see Table 'Supported values'. Unit: Hz

6.3.8.4.2.3 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.power.parameterSetList.filterPy.gauss.clone()
```

Subgroups

6.3.8.4.2.4 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.power.parameterSetList.filterPy.gauss.bandwidth.
↪get(index = 1)
```

Selects the bandwidth for a filter of Gaussian shape for the parameter set <Index>.

param index
integer Range: 0 to 31

return
bandwidth: numeric For supported values, see Table 'Supported values'. Unit: Hz

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth:ALL
value: List[float] = driver.configure.power.parameterSetList.filterPy.gauss.
↪bandwidth.get_all()
```

Selects the bandwidth for a filter of Gaussian shape for all parameter sets.

return
bandwidth: numeric Comma-separated list of 32 values, for parameter set 0 to 31 For supported values, see Table 'Supported values'. Unit: Hz

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth
driver.configure.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
integer Range: 0 to 31

param bandwidth
numeric For supported values, see Table 'Supported values'. Unit: Hz

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:FILTer:GAUSs:BWIDth:ALL
driver.configure.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
numeric Comma-separated list of 32 values, for parameter set 0 to 31 For supported values, see Table 'Supported values'. Unit: Hz

6.3.8.4.2.5 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.power.parameterSetList.
↪filterPy.typePy.get(index = 1)
```

Selects the IF filter type for the parameter set <Index>.

param index
integer Range: 0 to 31

return
filter_py: 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.power.parameterSetList.
↪filterPy.typePy.get_all()
```

Selects the IF filter type for all parameter sets.

return
filter_py: 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.power.parameterSetList.filterPy.typePy.set(index = 1, filter_
↪py = enums.DigitalFilterType.BANDpass)
```

Selects the IF filter type for the parameter set <Index>.

param index
integer Range: 0 to 31

param filter_py
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.power.parameterSetList.filterPy.typePy.set_all(filter_py =␣
↪[DigitalFilterType.BANDpass, DigitalFilterType.WCDMa])
```

Selects the IF filter type for all parameter sets.

param filter_py
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

6.3.8.4.3 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.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
integer Range: 0 to 31

return
meas_length: numeric Unit: s

`get_all`() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth:ALL
value: List[float] = driver.configure.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: numeric Comma-separated list of 32 values, for parameter set 0 to 31
Unit: s

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth
driver.configure.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
integer Range: 0 to 31

param meas_length
numeric Unit: s

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:MLENgth:ALL
driver.configure.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
numeric Comma-separated list of 32 values, for parameter set 0 to 31 Unit: s

6.3.8.4.4 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.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 RsCmwGprfMeas.Configure.Power.ParameterSetList. Catalog.pdefSet.

param index
integer Range: 0 to 31

return
predefined_set: string

set(*index: int*, *predefined_set: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:PDEFset
driver.configure.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 RsCmwGprfMeas.Configure.Power.ParameterSetList. Catalog.pdefSet.

param index
integer Range: 0 to 31

param predefined_set
string

6.3.8.4.5 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.power.parameterSetList.slength.get(index = 1)
```

Selects the time between the beginning of two consecutive measurement lengths for the parameter set <Index>.

param index
integer Range: 0 to 31

return
step_length: numeric Range: MeasLength to 1 s, Unit: s

get_all() → List[float]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth:ALL
value: List[float] = driver.configure.power.parameterSetList.slength.get_all()
```

Selects the time between the beginning of two consecutive measurement lengths for all parameter sets.

return
step_length: numeric Comma-separated list of 32 values, for parameter set 0 to 31 Range: MeasLength to 1 s, Unit: s

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth
driver.configure.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
integer Range: 0 to 31

param step_length
numeric Range: MeasLength to 1 s, Unit: s

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:PSET:SLENgth:ALL
driver.configure.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
numeric Comma-separated list of 32 values, for parameter set 0 to 31 Range: MeasLength to 1 s, Unit: s

6.3.8.5 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.power.trigger.get_source()
```

No command help available

return
source: No help available

set_source(*source: TriggerSource*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:TRIGger:SOURce
driver.configure.power.trigger.set_source(source = enums.TriggerSource.EXTernal)
```

No command help available

param source
No help available

6.3.9 RfSettings

SCPI Commands :

```
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. 8 total commands, 1 Subgroups, 7 group commands

get_eattenuation() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
value: float = driver.configure.rfSettings.get_eattenuation()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

return
rf_input_ext_att: numeric Range: -50 dB to 90 dB, Unit: dB

get_envelope_power() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:ENPower
value: float = driver.configure.rfSettings.get_envelope_power()
```

Sets the expected nominal power of the measured RF signal. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

return
exp_nominal_power: numeric 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 data sheet. Unit: dBm

get_foffset() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FOFFset
value: float = driver.configure.rfSettings.get_foffset()
```

Specifies a positive or negative frequency offset to be added to the center frequency (see method RsCmwGprfMeas.Configure. RfSettings.frequency) . This command does not apply to spectrum analysis in frequency sweep mode (see method RsCmwGprfMeas.Configure.Spectrum.Frequency.Span.mode) . This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

return
freq_offset: numeric Range: -100 kHz to 100 kHz, Unit: Hz

get_frequency() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
value: float = driver.configure.rfSettings.get_frequency()
```

Selects the center frequency of the RF analyzer. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command. For the supported frequency range, see 'Frequency ranges'.

return
analyzer_freq: numeric Unit: Hz

get_lr_interval() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRINterval
value: float = driver.configure.rfSettings.get_lr_interval()
```

No command help available

return
lvl_rang_interval: No help available

get_ml_offset() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:MLOFfset
value: float = driver.configure.rfSettings.get_ml_offset()
```

Varies the input level of the mixer in the analyzer path. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

return
mix_lev_offset: numeric Range: -10 dB to 16 dB, Unit: dB

get_umargin() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:UMARgin
value: float = driver.configure.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 data sheet. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

return
user_margin: numeric Range: 0 dB to (55 dB + external attenuation - expected nominal power) , Unit: dB

set_eattenuation(*rf_input_ext_att: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
driver.configure.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. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

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

set_envelope_power(*exp_nominal_power: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:ENPower
driver.configure.rfSettings.set_envelope_power(exp_nominal_power = 1.0)
```

Sets the expected nominal power of the measured RF signal. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

param exp_nominal_power
numeric 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 data sheet. Unit: dBm

set_foffset(*freq_offset: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FOFFset
driver.configure.rfSettings.set_foffset(freq_offset = 1.0)
```

Specifies a positive or negative frequency offset to be added to the center frequency (see method RsCmwGprfMeas.Configure. RfSettings.frequency) . This command does not apply to spectrum analysis in frequency sweep mode (see method RsCmwGprfMeas.Configure.Spectrum.Frequency.Span.mode) . This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

param freq_offset
numeric Range: -100 kHz to 100 kHz, Unit: Hz

set_frequency(*analyzer_freq: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
driver.configure.rfSettings.set_frequency(analyzer_freq = 1.0)
```

Selects the center frequency of the RF analyzer. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command. For the supported frequency range, see 'Frequency ranges'.

param analyzer_freq
numeric Unit: Hz

set_lr_interval(*lvl_rang_interval: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRINterval
driver.configure.rfSettings.set_lr_interval(lvl_rang_interval = 1.0)
```

No command help available

param lvl_rang_interval
No help available

set_ml_offset(*mix_lev_offset: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:MLOFfset
driver.configure.rfSettings.set_ml_offset(mix_lev_offset = 1.0)
```

Varies the input level of the mixer in the analyzer path. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

param mix_lev_offset
numeric Range: -10 dB to 16 dB, Unit: dB

set_umargin(*user_margin: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:UMARgin
driver.configure.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 data sheet. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command.

param user_margin
numeric Range: 0 dB to (55 dB + external attenuation - expected nominal power) , Unit: dB

Cloning the Group

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

Subgroups

6.3.9.1 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.rfSettings.lrStart.set()
```

No command help available

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

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:LRSTart
driver.configure.rfSettings.lrStart.set_with_opc()
```

No command help available

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

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

6.3.10 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. 22 total commands, 3 Subgroups, 4 group commands

get_amode() → AveragingMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
value: enums.AveragingMode = driver.configure.spectrum.get_amode()
```

Defines how the R&S CMW calculates the AVERage traces from the current results.

return
averaging_mode: LINear | LOGarithmic

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
value: enums.Repeat = driver.configure.spectrum.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
value: int = driver.configure.spectrum.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: numeric Number of measurement intervals Range: 1 to 1000

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
value: float = driver.configure.spectrum.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: numeric Unit: s

set_amode(*averaging_mode: AveragingMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
driver.configure.spectrum.set_amode(averaging_mode = enums.AveragingMode.LINear)
```

Defines how the R&S CMW calculates the AVERage traces from the current results.

param averaging_mode
LINear | LOGarithmic

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
driver.configure.spectrum.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 | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
driver.configure.spectrum.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
numeric Number of measurement intervals Range: 1 to 1000

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
driver.configure.spectrum.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
numeric Unit: s

Cloning the Group

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

Subgroups

6.3.10.1 FreqSweep

class FreqSweepCls

FreqSweep 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.configure.spectrum.freqSweep.clone()
```

Subgroups

6.3.10.1.1 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.spectrum.freqSweep.rbw.get_auto()
```

Enables or disables the automatic mode for the RBW in frequency sweep mode.

return
rbw_auto: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW
value: float = driver.configure.spectrum.freqSweep.rbw.get_value()
```

Configures the resolution bandwidth (RBW) for the frequency sweep mode. Setting this value is only possible if the automatic mode is off.

return
rbw: numeric Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 100 Hz to 10 MHz, Unit: Hz

set_auto(*rbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW:AUTO
driver.configure.spectrum.freqSweep.rbw.set_auto(rbw_auto = False)
```

Enables or disables the automatic mode for the RBW in frequency sweep mode.

param rbw_auto
OFF | ON

set_value(*rbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW
driver.configure.spectrum.freqSweep.rbw.set_value(rbw = 1.0)
```

Configures the resolution bandwidth (RBW) for the frequency sweep mode. Setting this value is only possible if the automatic mode is off.

param rbw
numeric Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 100 Hz to 10 MHz, Unit: Hz

6.3.10.1.2 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.spectrum.freqSweep.swt.get_auto()
```

Enables or disables the automatic mode for the sweep time in frequency sweep mode.

return
sweep_time_auto: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT
value: float = driver.configure.spectrum.freqSweep.swt.get_value()
```

Configures the sweep time for the frequency sweep mode. Setting this value is only possible if the automatic mode is off. The minimum allowed value depends on the span.

return
sweep_time: numeric Range: 0 s to 2000 s, Unit: s

set_auto(*sweep_time_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT:AUTO
driver.configure.spectrum.freqSweep.swt.set_auto(sweep_time_auto = False)
```

Enables or disables the automatic mode for the sweep time in frequency sweep mode.

param sweep_time_auto
OFF | ON

set_value(*sweep_time: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT
driver.configure.spectrum.freqSweep.swt.set_value(sweep_time = 1.0)
```

Configures the sweep time for the frequency sweep mode. Setting this value is only possible if the automatic mode is off. The minimum allowed value depends on the span.

param sweep_time
numeric Range: 0 s to 2000 s, Unit: s

6.3.10.1.3 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.spectrum.freqSweep.vbw.get_auto()
```

Enables or disables the automatic mode for the VBW in frequency sweep mode.

return
vbw_auto: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW
value: float or bool = driver.configure.spectrum.freqSweep.vbw.get_value()
```

Configures the video bandwidth (VBW) for the frequency sweep mode. Setting this value is only possible if the automatic mode is off.

return
vbw: (float or boolean) numeric | OFF Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 10 Hz to 10 MHz, Unit: Hz

set_auto(*vbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW:AUTO
driver.configure.spectrum.freqSweep.vbw.set_auto(vbw_auto = False)
```

Enables or disables the automatic mode for the VBW in frequency sweep mode.

param vbw_auto
OFF | ON

set_value(*vbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW
driver.configure.spectrum.freqSweep.vbw.set_value(vbw = 1.0)
```

Configures the video bandwidth (VBW) for the frequency sweep mode. Setting this value is only possible if the automatic mode is off.

param vbw
(float or boolean) numeric | OFF Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 10 Hz to 10 MHz, Unit: Hz

6.3.10.2 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.spectrum.frequency.get_center()
```

Configures the center frequency of the spectrum measurement. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command. For the supported frequency range, see 'Frequency ranges'.

return
center_frequency: numeric Unit: Hz

get_laspan() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:LASPan
value: float = driver.configure.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.spectrum.frequency.get_start()
```

Configures the start frequency of the frequency sweep. For the supported frequency range, see 'Frequency ranges'.

return
start_frequency: numeric Unit: Hz

get_stop() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
value: float = driver.configure.spectrum.frequency.get_stop()
```

Configures the stop frequency of the frequency sweep. For the supported frequency range, see 'Frequency ranges'.

return
stop_frequency: numeric Unit: Hz

set_center(*center_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:CENTer
driver.configure.spectrum.frequency.set_center(center_frequency = 1.0)
```

Configures the center frequency of the spectrum measurement. This command is only relevant for the standalone scenario. For the combined signal path scenario, use the corresponding ...:SIGN<i>:.. command. For the supported frequency range, see 'Frequency ranges'.

param center_frequency
numeric Unit: Hz

set_laspan(*last_span: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:LASPan
driver.configure.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.spectrum.frequency.set_start(start_frequency = 1.0)
```

Configures the start frequency of the frequency sweep. For the supported frequency range, see 'Frequency ranges'.

param start_frequency
numeric Unit: Hz

set_stop(*stop_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
driver.configure.spectrum.frequency.set_stop(stop_frequency = 1.0)
```

Configures the stop frequency of the frequency sweep. For the supported frequency range, see 'Frequency ranges'.

param stop_frequency
numeric Unit: Hz

Cloning the Group

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

Subgroups

6.3.10.2.1 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.spectrum.frequency.span.get_mode()
```

Configures the operating mode of the spectrum analyzer. FSWeep is not supported for combined signal path measurements.

return
span_mode: FSWeep | ZSPan FSWeep: frequency sweep mode ZSPan: zero span (time sweep) mode

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN
value: float = driver.configure.spectrum.frequency.span.get_value()
```

Configures the frequency span for frequency sweep mode. The supported frequency range depends on the instrument model and the available options. The supported range can be smaller than stated here. Refer to the preface of your model-specific base unit manual.

return
frequency_span: numeric Range: 1.0 kHz to 5.93 GHz, Unit: Hz

set_mode(*span_mode: SpanMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN:MODE
driver.configure.spectrum.frequency.span.set_mode(span_mode = enums.SpanMode.
↪FSWeep)
```

Configures the operating mode of the spectrum analyzer. FSWeep is not supported for combined signal path measurements.

param span_mode
FSWeep | ZSPan FSWeep: frequency sweep mode ZSPan: zero span (time sweep) mode

set_value(*frequency_span: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN
driver.configure.spectrum.frequency.span.set_value(frequency_span = 1.0)
```

Configures the frequency span for frequency sweep mode. The supported frequency range depends on the instrument model and the available options. The supported range can be smaller than stated here. Refer to the preface of your model-specific base unit manual.

param frequency_span
numeric Range: 1.0 kHz to 5.93 GHz, Unit: Hz

6.3.10.3 ZeroSpan

SCPI Command :

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
```

class ZeroSpanCls

ZeroSpan commands group definition. 6 total commands, 2 Subgroups, 1 group commands

get_swt() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
value: float = driver.configure.spectrum.zeroSpan.get_swt()
```

Configures the duration of one measurement interval for the zero span mode.

return
sweep_time: numeric Range: 0 s to 2000 s, Unit: s

set_swt(*sweep_time: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
driver.configure.spectrum.zeroSpan.set_swt(sweep_time = 1.0)
```

Configures the duration of one measurement interval for the zero span mode.

param sweep_time
numeric Range: 0 s to 2000 s, Unit: s

Cloning the Group

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

Subgroups

6.3.10.3.1 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.spectrum.zeroSpan.rbw.get_bandpass()
```

Configures the bandwidth of the bandpass resolution filter. In the current software version, the value is fixed.

return
rbw_bandpass: numeric Range: 40 MHz to 40 MHz, Unit: Hz

get_gauss() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:GAUSs
value: float = driver.configure.spectrum.zeroSpan.rbw.get_gauss()
```

Configures the bandwidth of the Gaussian resolution filter.

return
rbw: numeric Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 100 Hz to 10 MHz, Unit: Hz

get_type_py() → RbwFilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:TYPE
value: enums.RbwFilterType = driver.configure.spectrum.zeroSpan.rbw.get_type_
↪py()
```

Configures the type of the resolution filter to be used in zero span mode.

return
rbw_type: GAUSs | BANDpass

set_bandpass(*rbw_bandpass: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:BANDpass
driver.configure.spectrum.zeroSpan.rbw.set_bandpass(rbw_bandpass = 1.0)
```

Configures the bandwidth of the bandpass resolution filter. In the current software version, the value is fixed.

param rbw_bandpass
numeric Range: 40 MHz to 40 MHz, Unit: Hz

set_gauss(*rbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:GAUSs
driver.configure.spectrum.zeroSpan.rbw.set_gauss(rbw = 1.0)
```

Configures the bandwidth of the Gaussian resolution filter.

param rbw
numeric Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 100 Hz to 10 MHz, Unit: Hz

set_type_py(*rbw_type: RbwFilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:TYPE
driver.configure.spectrum.zeroSpan.rbw.set_type_py(rbw_type = enums.
↪RbwFilterType.BANDpass)
```

Configures the type of the resolution filter to be used in zero span mode.

param rbw_type
GAUSs | BANDpass

6.3.10.3.2 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.spectrum.zeroSpan.vbw.get_auto()
```

Enables or disables the automatic mode for the VBW in zero span mode.

return
vbw_auto: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW
value: float or bool = driver.configure.spectrum.zeroSpan.vbw.get_value()
```

Configures the video bandwidth (VBW) for the zero span mode. Setting this value is only possible if the automatic mode is off.

return
vbw: (float or boolean) numeric | OFF Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 10 Hz to 10 MHz, Unit: Hz

set_auto(*vbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW:AUTO
driver.configure.spectrum.zeroSpan.vbw.set_auto(vbw_auto = False)
```

Enables or disables the automatic mode for the VBW in zero span mode.

param vbw_auto
OFF | ON

set_value(*vbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW
driver.configure.spectrum.zeroSpan.vbw.set_value(vbw = 1.0)
```

Configures the video bandwidth (VBW) for the zero span mode. Setting this value is only possible if the automatic mode is off.

param vbw
(float or boolean) numeric | OFF Only certain values can be configured, see Table 'Supported values'. Other values are rounded to the next allowed value. Range: 10 Hz to 10 MHz, Unit: Hz

6.4 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: decimal See 'Reliability indicator'
- Current_Power: float: float Unit: dBm

- Average_Power: float: float Unit: dBm
- Minimum_Power: float: float Unit: dBm
- Maximum_Power: float: float Unit: dBm
- Elapsed_Stat: int: decimal Elapsed measurement cycles

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:EPSensor
driver.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.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.extPwrSensor.get_idn()
```

Returns the identification string of the connected external sensor.

return
idn: string

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:EPSensor
driver.extPwrSensor.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

```

(continues on next page)

(continued from previous page)

```
    - 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.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.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.extPwrSensor.clone()
```

Subgroups

6.4.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe
value: enums.ResourceState = driver.extPwrSensor.state.fetch(timeout = 1.0,␣
↪target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

Cloning the Group

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

Subgroups

6.4.1.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe:ALL
value: List[enums.ResourceState] = driver.extPwrSensor.state.all.fetch(timeout␣
↪= 1.0, target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.5 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.fftSpecAn.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:
```

(continues on next page)

(continued from previous page)

```
    - 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.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.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.fftSpecAn.clone()
```

Subgroups

6.5.1 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.fftSpecAn.icomponent.fetch()
```

Returns the measured normalized I and Q amplitudes in the time domain.

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

return
idata: float 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.fftSpecAn.icomponent.read()
```

Returns the measured normalized I and Q amplitudes in the time domain.

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

return
idata: float Comma-separated list of N normalized I or Q amplitudes. N equals the configured FFT length.

6.5.2 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.fftSpecAn.peaks.clone()
```

Subgroups

6.5.2.1 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: decimal See 'Reliability indicator'
- Frequency: List[float]: float Frequency of the detected peak Unit: Hz
- Level: List[float]: float Level of the detected peak Unit: dBm

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
value: ResultData = driver.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.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.

6.5.2.2 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: decimal See 'Reliability indicator'
- Frequency: List[float]: float Frequency of the detected peak Unit: Hz
- Level: List[float]: float Level of the detected peak Unit: dBm

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
value: ResultData = driver.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.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.

6.5.3 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.fftSpecAn.power.clone()
```

Subgroups

6.5.3.1 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.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:AVERage
value: List[float] = driver.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

6.5.3.2 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.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:CURRent
value: List[float] = driver.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

6.5.3.3 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.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MAXimum
value: List[float] = driver.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

6.5.3.4 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.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MINimum
value: List[float] = driver.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.

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

return
power: float Comma-separated list of 801 power values Unit: dBm

6.5.4 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.fftSpecAn.qcomponent.fetch()
```

Returns the measured normalized I and Q amplitudes in the time domain.

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

return
qdata: float 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.fftSpecAn.qcomponent.read()
```

Returns the measured normalized I and Q amplitudes in the time domain.

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

return
qdata: float Comma-separated list of N normalized I or Q amplitudes. N equals the configured FFT length.

6.5.5 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe
value: enums.ResourceState = driver.fftSpecAn.state.fetch(timeout = 1.0, target_
↪main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

Cloning the Group

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

Subgroups

6.5.5.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe:ALL
value: List[enums.ResourceState] = driver.fftSpecAn.state.all.fetch(timeout = 1.
↪0, target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.6 Initiate

class InitiateCls

Initiate 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.initiate.clone()
```

Subgroups

6.6.1 Ploss

class PlossCls

Ploss 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.initiate.ploss.clone()
```

Subgroups

6.6.1.1 Evaluate

SCPI Command :

```
INITiate:GPRF:MEASurement<Instance>:PLOSs:EVALuate
```

class EvaluateCls

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

class EvaluateStruct

Response structure. Fields:

- Connector: enums.CmwsConnector: No parameter help available
- Path_Index: enums.PathIndex: No parameter help available

get() → EvaluateStruct

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:EVALuate
value: EvaluateStruct = driver.initiate.ploss.evaluate.get()
```

No command help available

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

set(*connector: CmwsConnector*, *path_index: PathIndex = None*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:EVALuate
driver.initiate.ploss.evaluate.set(connector = enums.CmwsConnector.R11, path_
↪index = enums.PathIndex.P1)
```

No command help available

param connector
No help available

param path_index
No help available

6.6.1.2 Open

SCPI Command :

```
INITiate:GPRF:MEASurement<Instance>:PLOSs:OPEN
```

class OpenCls

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

class OpenStruct

Response structure. Fields:

- Connector: enums.CmwsConnector: No parameter help available
- Path_Index: enums.PathIndex: No parameter help available

get() → OpenStruct

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:OPEN
value: OpenStruct = driver.initiate.ploss.open.get()
```

No command help available

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

set(*connector: CmwsConnector*, *path_index: PathIndex = None*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:OPEN
driver.initiate.ploss.open.set(connector = enums.CmwsConnector.R11, path_index␣
↪= enums.PathIndex.P1)
```

No command help available

param connector
No help available

param path_index
No help available

6.6.1.3 Short

SCPI Command :

```
INITiate:GPRF:MEASurement<Instance>:PLOSs:SHORt
```

class ShortCls

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

class ShortStruct

Response structure. Fields:

- Connector: enums.CmwsConnector: No parameter help available
- Path_Index: enums.PathIndex: No parameter help available

get() → ShortStruct

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:SHORt
value: ShortStruct = driver.initiate.ploss.short.get()
```

No command help available

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

set(*connector: CmwsConnector*, *path_index: PathIndex = None*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:SHORt
driver.initiate.ploss.short.set(connector = enums.CmwsConnector.R11, path_index␣
↪= enums.PathIndex.P1)
```

No command help available

param connector
No help available

param path_index
No help available

6.7 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.iqRecorder.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
```

(continues on next page)

(continued from previous page)

```
    - 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.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 RsCmwGprfMeas.Configure.IqRecorder. Capture.set.

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

return
iq_samples: float 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. Unit: V (for ASCII format)

initiate(*save_to_iq_file: FileSave = None*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:IQRecorder
driver.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
OFF | ON | ONLY Optional parameter, selecting whether the results are written to an I/Q file, to the memory or both. For file selection, see method RsCmwGprfMeas.Configure.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.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 RsCmwGprfMeas.Configure.IqRecorder. Capture.set.

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

return
iq_samples: float 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. Unit: V (for ASCII format)

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:IQRecorder
driver.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.
    - 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.iqRecorder.clone()
```

Subgroups

6.7.1 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.iqRecorder.bin.fetch()
```

Returns I/Q recorder results in binary format. For the number of values n, see method RsCmwGprfMeas.Configure.IqRecorder. Capture.set.

return
iq_samples: block 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.iqRecorder.bin.read()
```

Returns I/Q recorder results in binary format. For the number of values n, see method RsCmwGprfMeas.Configure.IqRecorder. Capture.set.

return
iq_samples: block Binary block data. For a detailed description, see 'ASCII and binary data formats'.

6.7.2 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.iqRecorder.reliability.fetch()
```

Queries the reliability indicator for the I/Q recorder, see 'Reliability indicator'.

return
reliability_flag: decimal Two equal values, separated by a comma (e.g. 0,0 for 'OK')

6.7.3 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe
value: enums.ResourceState = driver.iqRecorder.state.fetch(timeout = 1.0,␣
↪target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

Cloning the Group

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

Subgroups

6.7.3.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe:ALL
value: List[enums.ResourceState] = driver.iqRecorder.state.all.fetch(timeout =␣
↪1.0, target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.7.4 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.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: float Unit: Hz

6.7.5 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.iqRecorder.talignment.fetch()
```

No command help available

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

return
time_alignment: No help available

read() → float

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQRecorder:TALignment
value: float = driver.iqRecorder.talignment.read()
```

No command help available

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

return
time_alignment: No help available

6.8 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.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.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.
```

(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.

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:IQVSlot
driver.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.
    - 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.iqVsSlot.clone()
```

Subgroups

6.8.1 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.iqVsSlot.freqError.fetch()
```

Returns the contents of the frequency error result diagram.

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

return
frequency_error: float Comma-separated list of frequency errors, one value per measured step Unit: Hz

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:FERRor
value: List[float] = driver.iqVsSlot.freqError.read()
```

Returns the contents of the frequency error result diagram.

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

return
frequency_error: float Comma-separated list of frequency errors, one value per measured step Unit: Hz

6.8.2 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.iqVsSlot.icomponent.fetch()
```

Returns the contents of the I and Q result diagrams.

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

return
idata: float Comma-separated list of amplitudes, one value per measured step Unit: V

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:I
value: List[float] = driver.iqVsSlot.icomponent.read()
```

Returns the contents of the I and Q result diagrams.

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

return
idata: float Comma-separated list of amplitudes, one value per measured step Unit: V

6.8.3 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.iqVsSlot.level.fetch()
```

Returns the contents of the level result diagram.

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

return
level: float Comma-separated list of levels, one value per measured step Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:LEVel
value: List[float] = driver.iqVsSlot.level.read()
```

Returns the contents of the level result diagram.

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

return
level: float Comma-separated list of levels, one value per measured step Unit: dBm

6.8.4 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.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.

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

return
frequency_error: float Overall frequency error, the arithmetic mean value of the frequency errors of all considered steps. Unit: Hz

fetch() → float

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:OFERror
value: float = driver.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.

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

return
frequency_error: float Overall frequency error, the arithmetic mean value of the frequency errors of all considered steps. Unit: Hz

read() → float

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:OFERror
value: float = driver.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.

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

return
frequency_error: float Overall frequency error, the arithmetic mean value of the frequency errors of all considered steps. Unit: Hz

6.8.5 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.iqVsSlot.phase.fetch()
```

Returns the contents of the phase result diagram.

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

return
phase: float Comma-separated list of phases, one value per measured step Unit: deg

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:PHASe
value: List[float] = driver.iqVsSlot.phase.read()
```

Returns the contents of the phase result diagram.

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

return
phase: float Comma-separated list of phases, one value per measured step Unit: deg

6.8.6 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.iqVsSlot.qcomponent.fetch()
```

Returns the contents of the I and Q result diagrams.

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

return
qdata: float Comma-separated list of amplitudes, one value per measured step Unit: V

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQVSlot:Q
value: List[float] = driver.iqVsSlot.qcomponent.read()
```

Returns the contents of the I and Q result diagrams.

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

return
qdata: float Comma-separated list of amplitudes, one value per measured step Unit: V

6.8.7 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:STATe
value: enums.ResourceState = driver.iqVsSlot.state.fetch(timeout = 1.0, target_
↪main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

Cloning the Group

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

Subgroups

6.8.7.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQVSlot:STATe:ALL
value: List[enums.ResourceState] = driver.iqVsSlot.state.all.fetch(timeout = 1.
↪0, target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.9 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.nrpm.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>:NRPM
driver.nrpm.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>:NRPM
driver.nrpm.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.nrpm.clone()
```

Subgroups

6.9.1 Sensor<Sensor>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.nrpm.sensor.repcap_sensor_get()
driver.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.nrpm.sensor.clone()
```

Subgroups

6.9.1.1 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: No parameter help available
- 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: No parameter help available
- Power_Antenna_1: float: No parameter help available
- Power_Antenna_2: float: No parameter help available
- Power_Antenna_3: float: No parameter help available

calculate(*sensor=Sensor.Default*) → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRPM:SENSor<nr_NRPM>:POWer
value: CalculateStruct = driver.nrpm.sensor.power.calculate(sensor = repcap.
↪Sensor.Default)
```

No command help available

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.nrpm.sensor.power.fetch(sensor = repcap.Sensor.
↪Default)
```

No command help available

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.nrpm.sensor.power.read(sensor = repcap.Sensor.
↪Default)
```

No command help available

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.

6.9.2 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRPM:STATe
value: enums.ResourceState = driver.nrpm.state.fetch(timeout = 1.0, target_main_
↪state = enums.TargetMainState.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.nrpm.state.clone()
```

Subgroups

6.9.2.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRPM:STATe:ALL
value: List[enums.ResourceState] = driver.nrpm.state.all.fetch(timeout = 1.0,␣
↪target_main_state = enums.TargetMainState.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

6.10 Ploss

SCPI Commands :

```
STOP:GPRF:MEASurement<Instance>:PLOSs
ABORt:GPRF:MEASurement<Instance>:PLOSs
```

class PlossCls

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

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

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:PLOSs
driver.ploss.abort()
```

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>:PLOSs
driver.ploss.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.ploss.clone()
```

Subgroups

6.10.1 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.ploss.clear.initiate()
```

No command help available

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

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:PLOSs:CLEar
driver.ploss.clear.initiate_with_opc()
```

No command help available

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

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

6.10.2 Eval

class EvalCls

Eval commands group definition. 5 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.10.2.1 Frequency

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:FREQuency
```

class FrequencyCls

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

fetch(*connector: CmwsConnector*, *path_index: PathIndex = None*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:FREQuency
value: List[float] = driver.ploss.eval.frequency.fetch(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

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

param connector
No help available

param path_index
No help available

return
frequency: No help available

6.10.2.2 Gain

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:GAIN
```

class GainCls

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

fetch(*connector: CmwsConnector*, *path_index: PathIndex = None*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:GAIN
value: List[float] = driver.ploss.eval.gain.fetch(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

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

param connector
No help available

param path_index
No help available

return
gain: No help available

6.10.2.3 State

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:STATe
```

class StateCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Result_State_Open: enums.PathLossState: No parameter help available
- Result_State_Short: enums.PathLossState: No parameter help available
- Result_State_Eval: enums.PathLossState: No parameter help available

fetch(*connector: CmwsConnector*, *path_index: PathIndex = None*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:STATe
value: FetchStruct = driver.ploss.eval.state.fetch(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

param connector
No help available

param path_index
No help available

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

6.10.2.4 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.ploss.eval.trace.clone()
```

Subgroups

6.10.2.4.1 Frequency

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:TRACe:FREQuency
```

class FrequencyCls

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

fetch(*connector: CmwsConnector*, *path_index: PathIndex = None*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:TRACe:FREQuency
value: List[float] = driver.ploss.eval.trace.frequency.fetch(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

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

param connector
No help available

param path_index
No help available

return
frequency: No help available

6.10.2.4.2 Gain

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:TRACe:GAIN
```

class GainCls

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

fetch(*connector: CmwsConnector*, *path_index: PathIndex = None*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:EVAL:TRACe:GAIN
value: List[float] = driver.ploss.eval.trace.gain.fetch(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

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

param connector
No help available

param path_index
No help available

return
gain: No help available

6.10.3 Open

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:OPEN
```

class OpenCls

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

fetch(*connector: CmwsConnector*, *path_index: PathIndex = None*) → PathLossState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:OPEN
value: enums.PathLossState = driver.ploss.open.fetch(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

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

param connector
No help available

param path_index
No help available

return
result_state_open: No help available

6.10.4 Short

SCPI Command :

```
FETCh:GPRF:MEASurement<Instance>:PLOSs:SHORt
```

class ShortCls

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

fetch(*connector: CmwsConnector*, *path_index: PathIndex = None*) → PathLossState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:SHORt
value: enums.PathLossState = driver.ploss.short.fetch(connector = enums.
↪CmwsConnector.R11, path_index = enums.PathIndex.P1)
```

No command help available

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

param connector
No help available

param path_index
No help available

return
result_state_short: No help available

6.10.5 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:STATe
value: enums.ResourceState = driver.ploss.state.fetch(timeout = 1.0, target_
↪main_state = enums.TargetMainState.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.ploss.state.clone()
```

Subgroups

6.10.5.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:PLOSs:STATe:ALL
value: List[enums.ResourceState] = driver.ploss.state.all.fetch(timeout = 1.0,␣
↪target_main_state = enums.TargetMainState.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

6.11 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.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.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.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.power.clone()
```

Subgroups

6.11.1 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.power.amplitudeProbDensity.fetch()
```

Returns the APD diagram contents.

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

return
results: float 4096 results, each representing a 0.047-dB interval (‘bin’) Unit: %

6.11.2 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.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.

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

return
power_average_rms: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[float] = driver.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.

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

return
power_average_rms: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[float] = driver.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.

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

return
power_average_rms: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

Cloning the Group

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

Subgroups

6.11.2.1 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.power.average.rms.fetch()
```

No command help available

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

return
power_average_rms: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:AVERage:RMS
value: List[float] = driver.power.average.rms.read()
```

No command help available

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

return
power_average_rms: No help available

6.11.3 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.power.cumulativeDistribFnc.fetch()
```

Returns the CCDF diagram contents.

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

return
results: float 4096 results, each representing a 0.047-dB interval (‘bin’) Unit: %

Cloning the Group

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

Subgroups

6.11.3.1 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: decimal See ‘Reliability indicator’
- Avg: float: float Average power of all samples Unit: dBm
- Max_Py: float: float Maximum power of all samples Unit: dBm
- Par: float: float Peak to average ratio Unit: dB
- Index_Avg_Power: int: decimal Index of the average power ‘bin’

fetch() → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:POWer
value: FetchStruct = driver.power.cumulativeDistribFnc.power.fetch()
```

Returns some statistic results for the power samples.

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

6.11.3.2 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.power.cumulativeDistribFnc.probability.fetch()
```

Returns power values with a certain probability, taken from the CCDF diagram.

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

return
probability: float 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. Unit: dB

6.11.3.3 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: decimal See 'Reliability indicator'
- Count: int: decimal Total sample count
- Time: float: float Total sample time Unit: s

fetch() → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CCDF:SAMPle
value: FetchStruct = driver.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.

6.11.4 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.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.

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

return
power_current_rms: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[float] = driver.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.

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

return
power_current_rms: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[float] = driver.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.

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

return
power_current_rms: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

Cloning the Group

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

Subgroups

6.11.4.1 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.power.current.maximum.fetch()
```

No command help available

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

return
power_current_max: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent:MAXimum
value: List[float] = driver.power.current.maximum.read()
```

No command help available

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

return
power_current_max: No help available

6.11.4.2 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.power.current.minimum.fetch()
```

No command help available

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

return
power_current_min: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent:MINimum
value: List[float] = driver.power.current.minimum.read()
```

No command help available

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

return
power_current_min: No help available

6.11.4.3 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.power.current.rms.fetch()
```

No command help available

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

return
power_current_rms: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent:RMS
value: List[float] = driver.power.current.rms.read()
```

No command help available

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

return
power_current_rms: No help available

6.11.5 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.power.elapsedStats.fetch()
```

Returns the reliability indicator and the number of elapsed measurement intervals.

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

return
stat_count: decimal Number of elapsed measurement intervals. Range: 0 to 100E+3

6.11.6 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.power.iqData.fetch(list_index = 1, result_index = 1)
```

No command help available

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

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.power.iqData.read(list_index = 1, result_index = 1)
```

No command help available

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

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.power.iqData.clone()
```

Subgroups

6.11.6.1 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.power.iqData.bin.fetch(list_index = 1, result_index␣
↪= 1)
```

No command help available

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

param list_index
No help available

param result_index
No help available

return
iq_data: No help available

6.11.7 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.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.

6.11.8 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.power.listPy.clone()
```

Subgroups

6.11.8.1 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.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.

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

param list_index
integer Index of the segment

return
power_average_rms: No help available

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_average_rms: No help available

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:AVERage
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_average_rms: No help available

6.11.8.2 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.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.

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

param list_index
integer Index of the segment

return
power_current_rms: No help available

`fetch`(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_current_rms: No help available

`read`(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:CURRent
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_current_rms: No help available

6.11.8.3 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.power.listPy.maximum.clone()
```

Subgroups

6.11.8.3.1 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.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.

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

param list_index
integer Index of the segment

return
power_current_max: No help available

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_current_max: No help available

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:MAXimum:CURRent
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_current_max: No help available

6.11.8.4 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.power.listPy.minimum.clone()
```

Subgroups

6.11.8.4.1 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.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.

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

param list_index
integer Index of the segment

return
power_current_min: No help available

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_current_min: No help available

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:MINimum:CURRent
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_current_min: No help available

6.11.8.5 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.power.listPy.peak.clone()
```

Subgroups

6.11.8.5.1 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.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.

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

param list_index
integer Index of the segment

return
power_maximum_max: No help available

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_maximum_max: No help available

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MAXimum
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_maximum_max: No help available

6.11.8.5.2 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.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.

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

param list_index
integer Index of the segment

return
power_minimum_min: No help available

`fetch`(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_minimum_min: No help available

`read`(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:PEAK:MINimum
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_minimum_min: No help available

6.11.8.6 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.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.

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

param list_index
integer Index of the segment

return
power_std_dev_cur: No help available

fetch(*list_index: int*) → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_std_dev_cur: No help available

read(*list_index: int*) → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:LIST:SDEViation
value: List[float] = driver.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.

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

param list_index
integer Index of the segment

return
power_std_dev_cur: No help available

6.11.9 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.power.maximum.clone()
```

Subgroups

6.11.9.1 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.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.

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

return

power_current_max: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[float] = driver.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.

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

return
power_current_max: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[float] = driver.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.

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

return
power_current_max: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

6.11.9.2 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.power.maximum.maximum.fetch()
```

No command help available

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

return
power_maximum_max: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MAXimum:MAXimum
value: List[float] = driver.power.maximum.maximum.read()
```

No command help available

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

return
power_maximum_max: No help available

6.11.10 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.power.minimum.clone()
```

Subgroups

6.11.10.1 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.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.

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

return
power_current_min: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[float] = driver.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.

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

return
power_current_min: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[float] = driver.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.

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

return
power_current_min: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

6.11.10.2 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.power.minimum.minimum.fetch()
```

No command help available

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

return
power_minimum_min: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MINimum:MINimum
value: List[float] = driver.power.minimum.minimum.read()
```

No command help available

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

return
power_minimum_min: No help available

6.11.11 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.power.peak.clone()
```

Subgroups

6.11.11.1 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.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.

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

return
power_maximum_max: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[float] = driver.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.

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

return
power_maximum_max: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[float] = driver.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.

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

return
power_maximum_max: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

6.11.11.2 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.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.

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

return
power_minimum_min: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[float] = driver.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.

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

return
power_minimum_min: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[float] = driver.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.

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

return
power_minimum_min: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

6.11.12 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.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.

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

return
power_std_dev_cur: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[float] = driver.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.

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

return
power_std_dev_cur: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[float] = driver.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.

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

return
power_std_dev_cur: float Comma-separated list of power values, one value per measured segment Unit: dBm (SDEViation: dB)

Cloning the Group

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

Subgroups

6.11.12.1 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.power.standardDev.current.fetch()
```

No command help available

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

return
power_std_dev_cur: No help available

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:SDEViation:CURRent
value: List[float] = driver.power.standardDev.current.read()
```

No command help available

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

return
power_std_dev_cur: No help available

6.11.13 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:STATe
value: enums.ResourceState = driver.power.state.fetch(timeout = 1.0, target_
↪main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

Cloning the Group

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

Subgroups

6.11.13.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:STATe:ALL
value: List[enums.ResourceState] = driver.power.state.all.fetch(timeout = 1.0,␣
↪target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.12 Route

SCPI Command :

```
ROUTe:GPRF:MEASurement<Instance>
```

class RouteCls

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

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SALone | CSPath SALone: standalone (non-signaling) scenario CSPath: combined signal path scenario
- Master: str: string Controlling application for scenario CSPath
- Rf_Connector: enums.RfConnector: RF connector for the input path
- Rf_Converter: enums.RxConverter: RX module for the input path

get_value() → ValueStruct

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>
value: ValueStruct = driver.route.get_value()
```

Returns the configured routing settings. For possible connector and converter values, see 'Values for RF path selection'.

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

Cloning the Group

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

Subgroups

6.12.1 Scenario

SCPI Commands :

```
ROUTe:GPRF:MEASurement<Instance>:SCENario:CSPath
ROUTe:GPRF:MEASurement<Instance>:SCENario
```

class ScenarioCls

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

get_cspath() → str

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:CSPath
value: str = driver.route.scenario.get_cspath()
```

Activates the combined signal path scenario and selects a controlling firmware application for the GPRF measurements. The selected application controls the signal routing settings and analyzer settings while the combined signal path scenario is active. To query a list of possible <Controller> values, see method RsCmwGprfMeas.Route.Scenario.Catalog.cspath.

return
master: No help available

get_value() → Scenario

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario
value: enums.Scenario = driver.route.scenario.get_value()
```

Queries the active scenario.

return
scenario: SALone | CSPath SALone: standalone (non-signaling) scenario CSPath: combined signal path scenario

set_cspath(*master: str*) → None

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:CSPath
driver.route.scenario.set_cspath(master = 'abc')
```

Activates the combined signal path scenario and selects a controlling firmware application for the GPRF measurements. The selected application controls the signal routing settings and analyzer settings while the combined signal path scenario is active. To query a list of possible <Controller> values, see method RsCmwGprfMeas.Route.Scenario.Catalog.cspath.

param master
string Example: 'LTE Sig1'

Cloning the Group

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

Subgroups

6.12.1.1 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.scenario.catalog.get_cspath()
```

Queries the possible controlling firmware applications for a combined signal path scenario. The returned list contains all installed signaling applications.

return
csp_masters: string Comma-separated list of string parameters For example 'LTE Sig1', 'GSM Sig2' 'No Connection' means that no signaling application is installed.

6.12.1.2 Maiq

SCPI Command :

```
ROUTe:GPRF:MEASurement<Instance>:SCENario:MAIQ
```

class MaiqCls

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

class MaiqStruct

Response structure. Fields:

- Rx_Connector: enums.RfConnector: No parameter help available
- Rf_Converter: enums.RxConverter: No parameter help available

get() → MaiqStruct

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:MAIQ
value: MaiqStruct = driver.route.scenario.maiq.get()
```

No command help available

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

set(*rx_connector: RfConnector*, *rf_converter: RxConverter*) → None

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:MAIQ
driver.route.scenario.maiq.set(rx_connector = enums.RfConnector.I11I, rf_
↪converter = enums.RxConverter.IRX1)
```

No command help available

param rx_connector
No help available

param rf_converter
No help available

6.12.1.3 MaProtocol

SCPI Command :

```
ROUTe:GPRF:MEASurement<Instance>:SCENario:MAPRotocol
```

class MaProtocolCls

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

set(*controler: str = None*, *converter: RxConverter = None*) → None

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:MAPRotocol
driver.route.scenario.maProtocol.set(controler = 'abc', converter = enums.
↪RxConverter.IRX1)
```

No command help available

param controler
No help available

param converter
No help available

6.12.1.4 Salone

SCPI Command :

```
ROUTe:GPRF:MEASurement<Instance>:SCENario:SALone
```

class SaloneCls

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

class SaloneStruct

Response structure. Fields:

- Rx_Connector: enums.RfConnector: RF connector for the input path
- Rf_Converter: enums.RxConverter: RX module for the input path

get() → SaloneStruct

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:SALone
value: SaloneStruct = driver.route.scenario.salone.get()
```

Activates the standalone scenario and selects the RF input path for the measured RF signal. For possible connector and converter values, see 'Values for RF path selection'.

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

set(*rx_connector: RfConnector*, *rf_converter: RxConverter*) → None

```
# SCPI: ROUTe:GPRF:MEASurement<Instance>:SCENario:SALone
driver.route.scenario.salone.set(rx_connector = enums.RfConnector.I11I, rf_
↪converter = enums.RxConverter.IRX1)
```

Activates the standalone scenario and selects the RF input path for the measured RF signal. For possible connector and converter values, see 'Values for RF path selection'.

param rx_connector
RF connector for the input path

param rf_converter
RX module for the input path

6.13 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.spectrum.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>:SPECtrum
driver.spectrum.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
```

(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.

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

```
# SCPI: STOP:GPRF:MEASurement<Instance>:SPECtrum
driver.spectrum.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.spectrum.clone()
```

Subgroups

6.13.1 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.spectrum.average.clone()
```

Subgroups

6.13.1.1 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.spectrum.average.average.fetch()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:AVERage
value: List[float] = driver.spectrum.average.average.read()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.1.2 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.spectrum.average.current.fetch()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:CURRent
value: List[float] = driver.spectrum.average.current.read()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.1.3 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.spectrum.average.maximum.fetch()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MAXimum
value: List[float] = driver.spectrum.average.maximum.read()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.1.4 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.spectrum.average.minimum.fetch()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MINimum
value: List[float] = driver.spectrum.average.minimum.read()
```

Returns the traces calculated with the average detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.2 Marker<Marker>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.spectrum.marker.repcap_marker_get()
driver.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.spectrum.marker.clone()
```

Subgroups

6.13.2.1 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: decimal See 'Reliability indicator'
- Xvalue: float: float X value
- Yvalue: float: float Y value Unit: dBm

fetch(*detector: Detector*, *statistic: Statistic*, *marker=Marker.Default*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:MARKer<MarkerNo>:NPEak
value: FetchStruct = driver.spectrum.marker.npeak.fetch(detector = enums.
↪Detector.AUTopeak, statistic = enums.Statistic.AVERage, marker = repcap.
↪Marker.Default)
```

Moves marker <MarkerNo> to the next lower (or equal) peak, relative to the current marker position. Returns the X and Y value of the new marker position. The trace is selected by <Detector> and <Statistic>.

param detector
AVERage | RMS | SAMPle | MINPeak | MAXPeak | AUTopeak Selects the detector type, see 'Detector hotkey'.

param statistic
CURRent | AVERage | MAXimum | MINimum Selects the trace type.

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.

6.13.3 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.spectrum.maximum.clone()
```

Subgroups

6.13.3.1 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.spectrum.maximum.average.fetch()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:AVERage
value: List[float] = driver.spectrum.maximum.average.read()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.3.2 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.spectrum.maximum.current.fetch()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:CURRent
value: List[float] = driver.spectrum.maximum.current.read()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.3.3 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.spectrum.maximum.maximum.fetch()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MAXimum
value: List[float] = driver.spectrum.maximum.maximum.read()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.3.4 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.spectrum.maximum.minimum.fetch()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MINimum
value: List[float] = driver.spectrum.maximum.minimum.read()
```

Returns the traces calculated with the maximum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.4 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.spectrum.minimum.clone()
```

Subgroups

6.13.4.1 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.spectrum.minimum.average.fetch()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:AVERage
value: List[float] = driver.spectrum.minimum.average.read()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.4.2 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.spectrum.minimum.current.fetch()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:CURRent
value: List[float] = driver.spectrum.minimum.current.read()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.4.3 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.spectrum.minimum.maximum.fetch()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MAXimum
value: List[float] = driver.spectrum.minimum.maximum.read()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.4.4 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.spectrum.minimum.minimum.fetch()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MINimum
value: List[float] = driver.spectrum.minimum.minimum.read()
```

Returns the traces calculated with the minimum peak detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.5 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.spectrum.referenceMarker.clone()
```

Subgroups

6.13.5.1 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: decimal See 'Reliability indicator'
- Xvalue: float: float X value
- Yvalue: float: float Y value Unit: dBm

fetch(*detector: Detector*, *statistic: Statistic*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:NPEak
value: FetchStruct = driver.spectrum.referenceMarker.npeak.fetch(detector =␣
↪enums.Detector.AUTopeak, statistic = enums.Statistic.AVERage)
```

Moves the reference marker to the next lower (or equal) peak, relative to the current marker position. Returns the X and Y value of the new marker position. The trace is selected by <Detector> and <Statistic>.

param detector
AVERage | RMS | SAMPle | MINPeak | MAXPeak | AUTopeak Selects the detector type, see 'Detector hotkey'.

param statistic
CURRent | AVERage | MAXimum | MINimum Selects the trace type.

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

6.13.5.2 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: decimal See 'Reliability indicator'
- Xvalue: float: float X value
- Yvalue: float: float Y value Unit: dBm

fetch(*detector: Detector*, *statistic: Statistic*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:SPEak
value: FetchStruct = driver.spectrum.referenceMarker.speak.fetch(detector =␣
↪enums.Detector.AUTopeak, statistic = enums.Statistic.AVERage)
```

Moves the reference marker to the highest peak of the trace determined by <Detector> and <Statistic> and returns the X and Y value of the new marker position.

param detector
AVERage | RMS | SAMPle | MINPeak | MAXPeak | AUTopeak Selects the detector type, see 'Detector hotkey'.

param statistic
CURRent | AVERage | MAXimum | MINimum Selects the trace type.

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

6.13.6 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.spectrum.rms.clone()
```

Subgroups

6.13.6.1 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.spectrum.rms.average.fetch()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:AVERage
value: List[float] = driver.spectrum.rms.average.read()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.6.2 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.spectrum.rms.current.fetch()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:CURRent
value: List[float] = driver.spectrum.rms.current.read()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.6.3 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.spectrum.rms.maximum.fetch()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MAXimum
value: List[float] = driver.spectrum.rms.maximum.read()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.6.4 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.spectrum.rms.minimum.fetch()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MINimum
value: List[float] = driver.spectrum.rms.minimum.read()
```

Returns the traces calculated with the RMS detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.7 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.spectrum.sample.clone()
```

Subgroups

6.13.7.1 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.spectrum.sample.average.fetch()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:AVERage
value: List[float] = driver.spectrum.sample.average.read()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.7.2 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.spectrum.sample.current.fetch()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:CURRent
value: List[float] = driver.spectrum.sample.current.read()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.7.3 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.spectrum.sample.maximum.fetch()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MAXimum
value: List[float] = driver.spectrum.sample.maximum.read()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.7.4 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.spectrum.sample.minimum.fetch()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MINimum
value: List[float] = driver.spectrum.sample.minimum.read()
```

Returns the traces calculated with the sample detector. Current, average, maximum and minimum traces can be retrieved.

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

return
power: float Comma-separated list of 1001 values Unit: dBm

6.13.8 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe
value: enums.ResourceState = driver.spectrum.state.fetch(timeout = 1.0, target_
↪main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

Cloning the Group

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

Subgroups

6.13.8.1 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: TargetMainState = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe:ALL
value: List[enums.ResourceState] = driver.spectrum.state.all.fetch(timeout = 1.
↪0, target_main_state = enums.TargetMainState.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
numeric Unit: ms

param target_main_state
OFF | RUN | RDY Target MainState for the query Default is RUN.

param target_sync_state
PENDing | ADJusted Target SyncState for the query Default is ADJ.

return
meas_state: No help available

6.14 Trigger

class TriggerCls

Trigger commands group definition. 43 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.14.1 FftSpecAn

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
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
```

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.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: numeric Range: 0 s to 0.01 s, Unit: s

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OFFSet
value: float = driver.trigger.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: numeric Range: -0.15 s to 0.15 s, Unit: s

get_omode() → OffsetMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
value: enums.OffsetMode = driver.trigger.fftSpecAn.get_omode()
```

Selects the trigger offset mode.

return
offset_mode: VARiable | FIXed

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SLOPe
value: enums.SignalSlopeExt = driver.trigger.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 | FEDGe REDGe: rising edge FEDGe: falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
value: str = driver.trigger.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
source: string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:THReshold
value: float = driver.trigger.fftSpecAn.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
value: float or bool = driver.trigger.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) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:MGAP
driver.trigger.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
numeric Range: 0 s to 0.01 s, Unit: s

set_offset(*offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OFFSet
driver.trigger.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
numeric Range: -0.15 s to 0.15 s, Unit: s

set_omode(*offset_mode: OffsetMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
driver.trigger.fftSpecAn.set_omode(offset_mode = enums.OffsetMode.FIXed)
```

Selects the trigger offset mode.

param offset_mode
VARiable | FIXed

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SLOPe
driver.trigger.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 | FEDGe REDGe: rising edge FEDGe: falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
driver.trigger.fftSpecAn.set_source(source = '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 source
string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:THReshold
driver.trigger.fftSpecAn.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
driver.trigger.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) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

Cloning the Group

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

Subgroups

6.14.1.1 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.fftSpecAn.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCmwGprfMeas.Trigger.FftSpecAn.source.

return
trigger_sources: string Comma-separated list of all supported values. Each value is represented as a string.

6.14.1.2 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: numeric Range: -0.15 s to 0.15 s, Unit: s
- Offset_Stop: float: numeric Range: -0.15 s to 0.15 s, Unit: s

get() → OsStopStruct

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OSSTop
value: OsStopStruct = driver.trigger.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.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
numeric Range: -0.15 s to 0.15 s, Unit: s

param offset_stop
numeric Range: -0.15 s to 0.15 s, Unit: s

6.14.2 IqRecorder

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
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
```

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.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 runs always 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: numeric Range: 0 s to 0.01 s, Unit: s

get_offset() → int

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
value: int = driver.trigger.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: numeric Trigger offset in samples. Range: 0 to 64E+6

get_pc_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTHreshold
value: float = driver.trigger.iqRecorder.get_pc_threshold()
```

No command help available

return
phase_chg_thres: No help available

get_pc_time() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTime
value: float or bool = driver.trigger.iqRecorder.get_pc_time()
```

No command help available

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.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 | FEDGe REDGe: rising edge FEDGe: falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
value: str = driver.trigger.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
source: string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
value: float = driver.trigger.iqRecorder.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
value: float or bool = driver.trigger.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) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:MGAP
driver.trigger.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 runs always 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
numeric Range: 0 s to 0.01 s, Unit: s

set_offset(*trigger_offset: int*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
driver.trigger.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
numeric Trigger offset in samples. Range: 0 to 64E+6

set_pc_threshold(*phase_chg_thres: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTHreshold
driver.trigger.iqRecorder.set_pc_threshold(phase_chg_thres = 1.0)
```

No command help available

param phase_chg_thres
No help available

set_pc_time(*phase_chg_time: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:PCTime
driver.trigger.iqRecorder.set_pc_time(phase_chg_time = 1.0)
```

No command help available

param phase_chg_time
(float or boolean) No help available

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SLOPe
driver.trigger.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 | FEDGe REDGe: rising edge FEDGe: falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
driver.trigger.iqRecorder.set_source(source = '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 source
string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
driver.trigger.iqRecorder.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
driver.trigger.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) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

Cloning the Group

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

Subgroups

6.14.2.1 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.iqRecorder.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCmwGprfMeas.Trigger.IqRecorder.source.

return
trigger_sources: string Comma-separated list of all supported values. Each value is represented as a string.

6.14.3 IqVsSlot

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SOURce
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
```

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.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: numeric Range: 0 s to 0.01 s, Unit: s

get_mode() → TriggerSequenceMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MODE
value: enums.TriggerSequenceMode = driver.trigger.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 | PRESelect ONCE: 'Trigger Once' PRESelect: 'Retrigger Preselect'

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:OFFSet
value: float = driver.trigger.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: numeric Range: 0 s to 1 s, Unit: s

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SLOPe
value: enums.SignalSlopeExt = driver.trigger.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 | FEDGe REDGe: Rising edge FEDGe: Falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SOURce
value: str = driver.trigger.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
source: string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:THReshold
value: float = driver.trigger.iqVsSlot.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:TOUT
value: float or bool = driver.trigger.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) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MGAP
driver.trigger.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
numeric Range: 0 s to 0.01 s, Unit: s

set_mode(*mode: TriggerSequenceMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:MODE
driver.trigger.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 | PRESelect ONCE: 'Trigger Once' PRESelect: 'Retrigger Preselect'

set_offset(*offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:OFFSet
driver.trigger.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
numeric Range: 0 s to 1 s, Unit: s

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SLOPe
driver.trigger.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 | FEDGe REDGe: Rising edge FEDGe: Falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:SOURce
driver.trigger.iqVsSlot.set_source(source = '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 source
string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:THReshold
driver.trigger.iqVsSlot.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQVSlot:TOUT
driver.trigger.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) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

Cloning the Group

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

Subgroups

6.14.3.1 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.iqVsSlot.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCmwGprfMeas.Trigger.IqVsSlot.source.

return
trigger_sources: string Comma-separated list of all supported values. Each value is represented as a string.

6.14.4 Power

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
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
```

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.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: numeric Range: 0 s to 0.01 s, Unit: s

get_mode() → TriggerPowerMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MODE
value: enums.TriggerPowerMode = driver.trigger.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 | SWEep | ALL | PRESelect 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.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: numeric Range: 0 s to 1 s, Unit: s

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SLOPe
value: enums.SignalSlopeExt = driver.trigger.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 | FEDGe REDGe: rising edge FEDGe: falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
value: str = driver.trigger.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
source: string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:THReshold
value: float = driver.trigger.power.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:TOUT
value: float or bool = driver.trigger.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) Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MGAP
driver.trigger.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
numeric Range: 0 s to 0.01 s, Unit: s

set_mode(*mode: TriggerPowerMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MODE
driver.trigger.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 | SWEep | ALL | PRESelect 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.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
numeric Range: 0 s to 1 s, Unit: s

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SLOPe
driver.trigger.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 | FEDGe REDGe: rising edge FEDGe: falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
driver.trigger.power.set_source(source = '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 source
string 'IF Power': IF power trigger 'Free Run': free run (untriggered)

`set_threshold`(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:THReshold
driver.trigger.power.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

`set_timeout`(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:TOUT
driver.trigger.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) Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

Cloning the Group

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

Subgroups

6.14.4.1 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.power.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCmwGprfMeas.Trigger.Power.source.

return
trigger_sources: string Comma-separated list of all supported values. Each value is represented as a string.

6.14.4.2 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.power.parameterSetList.clone()
```

Subgroups

6.14.4.2.1 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.power.parameterSetList.offset.get(index = 1)
```

Defines a delay time relative to the trigger event for the parameter set <Index>.

param index
integer Range: 0 to 31

return
trigger_offset: numeric Range: 0 s to 1 s, Unit: s

get_all() → List[float]

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet:ALL
value: List[float] = driver.trigger.power.parameterSetList.offset.get_all()
```

Defines a delay time relative to the trigger event for all parameter sets.

return
trigger_offset: numeric Comma-separated list of 32 offsets, for parameter set 0 to 31 Range: 0 s to 1 s, Unit: s

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

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet
driver.trigger.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
integer Range: 0 to 31

param trigger_offset
numeric Range: 0 s to 1 s, Unit: s

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

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:PSET:OFFSet:ALL
driver.trigger.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
numeric Comma-separated list of 32 offsets, for parameter set 0 to 31 Range: 0 s to 1 s, Unit: s

6.14.5 Spectrum

SCPI Commands :

```
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SOURce
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. 7 total commands, 1 Subgroups, 6 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:MGAP
value: float = driver.trigger.spectrum.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: numeric Range: 0 s to 0.01 s, Unit: s

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:OFFSet
value: float = driver.trigger.spectrum.get_offset()
```

Defines the trigger offset, i.e. the offset of a triggered zero span measurement relative to the corresponding trigger event.

return
trigger_offset: numeric Range: -0.5 s to 0.5 s, Unit: s

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SLOPe
value: enums.SignalSlopeExt = driver.trigger.spectrum.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
slope: REDGe | FEDGe REDGe: rising edge FEDGe: falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SOURce
value: str = driver.trigger.spectrum.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
source: string 'Free Run': free run (untriggered) 'Video': power trigger at video band

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
value: float = driver.trigger.spectrum.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
threshold: numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:TOUT
value: float or bool = driver.trigger.spectrum.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
trigger_timeout: (float or boolean) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:MGAP
driver.trigger.spectrum.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
numeric Range: 0 s to 0.01 s, Unit: s

`set_offset`(*trigger_offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:OFFSet
driver.trigger.spectrum.set_offset(trigger_offset = 1.0)
```

Defines the trigger offset, i.e. the offset of a triggered zero span measurement relative to the corresponding trigger event.

param trigger_offset
numeric Range: -0.5 s to 0.5 s, Unit: s

`set_slope`(*slope: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SLOPe
driver.trigger.spectrum.set_slope(slope = 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 slope
REDGe | FEDGe REDGe: rising edge FEDGe: falling edge

`set_source`(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SOURce
driver.trigger.spectrum.set_source(source = '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 source
string 'Free Run': free run (untriggered) 'Video': power trigger at video band

`set_threshold`(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
driver.trigger.spectrum.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param threshold
numeric Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to reference level minus external attenuation)

`set_timeout`(*trigger_timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:TOUT
driver.trigger.spectrum.set_timeout(trigger_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 trigger_timeout
(float or boolean) numeric | ON | OFF Range: 0.01 s to 300 s, Unit: s ON | OFF enables or disables the timeout check.

Cloning the Group

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

Subgroups

6.14.5.1 Catalog

SCPI Command :

```
TRIGger:GPRF:MEASurement<Instance>:SPECtrum: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>:SPECtrum:CATalog:SOURce
value: List[str] = driver.trigger.spectrum.catalog.get_source()
```

Lists all trigger source values that can be set using method RsCmwGprfMeas.Trigger.Spectrum.source.

return
trigger_sources: string Comma-separated list of all supported values. Each value is represented as a string.

CHAPTER

SEVEN

RSCMWGPRFMEAS UTILITIES

class Utilities

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

Access snippet: `utils = RsCmwGprfMeas.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

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

RSCMWGPRFMEAS 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

RSCMWGPRFMEAS EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

TEN

INDEX

INDEX

D

E

F

G

I

L

M

R

S

T

U