RsCma

Release 4.0.10.30

Rohde & Schwarz

Jul 11, 2023

CONTENTS:

CONTENTS:

CHAPTER

ONE

REVISION HISTORY

1.1 RsCma

Rohde & Schwarz CMA180 Radio Tester Driver RsCma instrument driver.

Basic Hello-World code:

```
from RsCma import *

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

Supported instruments: CMA180

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

Documentation: https://RsCma.readthedocs.io/

Examples: https://github.com/Rohde-Schwarz/Examples/tree/main/RadioTestSets/Python/RsCma_ScpiPackage

1.1.1 Version history

Latest release notes summary: Update for Firmware 4.0.10

Version 4.0.10

- Update for Firmware 4.0.10

Version 1.7.20

- Update for Firmware 1.7.20

Version 1.7.10

- Fixed bug in interfaces with the name 'base'

Version 1.7.10

- Update for Firmware 1.7.10
- Fixed several misspelled errors

Version 1.5.70

- Added documentation on ReadTheDocs

Version 1.5.70.21

- Added new data types for commands accepting numbers or ON/OFF:
- int or bool
- float or bool

Version 1.5.70.0018

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

RsCma 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 RsCma example:

```
# Example of using R&S CMA180 auto-generated instrument driver module RsCma for Python 3

from RsCma import *  # install from pypi.org

RsCma.assert_minimum_version('1.5.70')
cma = RsCma('TCPIP::10.112.1.116::INSTR', True, False)
print(f'CMA Identification: {cma.utilities.idn_string}')

# SYSTem:DISPlay:UPDate
cma.system.display.set_update(False)

# SOURce:AFRF:GEN:STATe
cma.source.afRf.generator.state.set(True)

# SOURce:AFRF:GEN:RFSettings:FREQuency
cma.source.afRf.generator.rfSettings.set_frequency(100E3)
```

(continues on next page)

(continued from previous page)

```
# SOURce:AFRF:GEN:RFSettings:LEVel
cma.source.afRf.generator.rfSettings.set_level(-20.0)

# CONFigure:GPRF:MEAS:SPECtrum:FREQuency:SPAN:MODE
cma.configure.gprfMeasurement.spectrum.frequency.span.set_mode(enums.SpanMode.FSWeep)

# CONFigure:GPRF:MEAS:SPECtrum:SCOunt
cma.configure.gprfMeasurement.spectrum.set_scount(10)

# CONFigure:GPRF:MEAS:SPECtrum:REPetition
cma.configure.gprfMeasurement.spectrum.set_repetition(enums.Repeat.SINGleshot)

# CONFigure:GPRF:MEAS:SPECtrum:FREQuency:SPAN
cma.configure.gprfMeasurement.spectrum.frequency.span.set_value(1.1E6)

# CONFigure:GPRF:MEAS:SPECtrum:FREQuency:CENTer
cma.configure.gprfMeasurement.spectrum.frequency.set_center(100E3)

# INITiate:GPRF:MEAS:SPECtrum
cma.gprfMeasurement.spectrum.initiate()
cma.gprfMeasurement.spectrum.initiate_with_opc()

# READ:GPRF:MEAS:POWer:CURRent?
results_power = cma.gprfMeasurement.power.current.read()

# FETCh:GPRF:MEAS:SPECtrum:RMS:CURRent?
results_spectrum = cma.gprfMeasurement.spectrum.rms.current.fetch()

# SOURce:AFRF:GENerator:IGENerator<nr>:MTONe:TONE<no>:ENABle
# option A: stating all the repcaps in the method call
# sends: SOURce:AFRF:GENerator:IGENerator2:MTONe:TONE4:ENABle ON
cma.source.afRf.generator.internalGenerator.multiTone.tone.enable.set(True, repcap.
↪InternalGen.Nr2, repcap.ToneNumber.Nr4)

# option B - setting default values of the repcaps in the groups:
cma.source.afRf.generator.internalGenerator.repcap_internalGen_set(repcap.InternalGen.
↪Nr2)
cma.source.afRf.generator.internalGenerator.multiTone.tone.repcap_toneNumber_set(repcap.
↪ToneNumber.Nr4)

# Then use the method call without defining the repcaps - they stay at the default value
# sends: SOURce:AFRF:GENerator:IGENerator2:MTONe:TONE4:ENABle ON
cma.source.afRf.generator.internalGenerator.multiTone.tone.enable.set(True)

# cloning instances:
igen3 = cma.source.afRf.generator.internalGenerator.clone()
igen3.repcap_internalGen_set(repcap.InternalGen.Nr3)
igen3.multiTone.tone.enable.set(True)
# sends: SOURce:AFRF:GENerator:IGENerator3:MTONe:TONE4:ENABle ON

igen3.multiTone.tone.repcap_toneNumber_set(repcap.ToneNumber.Nr7)
# sends: SOURce:AFRF:GENerator:IGENerator3:MTONe:TONE7:ENABle ON
```

(continues on next page)

(continued from previous page)

```
cma.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

RsCma 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 RsCma`

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 `RsCma` 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 RsCma offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCma needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCma package to your computer from the pypi.org: https://pypi.org/project/RsCma/#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\RsCma-4.0.10.30.tar`

2.3 Finding Available Instruments

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

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

from RsCma import *

# Use the instr_list string items as resource names in the RsCma constructor
instr_list = RsCma.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
"""
```

(continues on next page)

(continued from previous page)

```
from RsCma 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 = RsCma.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

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

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

- Installed RsCma Python module Version 4.0.10 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCma import *

# A good practice is to assure that you have a certain minimum version installed
RsCma.assert_minimum_version('4.0.10')
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
```

(continues on next page)

(continued from previous page)

```
driver = RsCma(resource_string_1)

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

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

from RsCma import *

# Force use of the Rs Visa. For NI Visa, use the "SelectVisa='ni'"
driver = RsCma('TCPIP::192.168.56.101::INSTR', True, True, "SelectVisa='rs'")

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

(continues on next page)

(continued from previous page)

```
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, RsCma has something for you too - **no Visa installation raw LAN socket**:

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

from RsCma import *

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

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

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

Simulating Session

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

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

More `option_string` tokens are separated by comma:

```
driver = RsCma('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 RsCma objects:

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

from RsCma import *
```

(continues on next page)

(continued from previous page)

```
driver1 = RsCma('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsCma.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 RsCma 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
"""
```

(continues on next page)

(continued from previous page)

```
from RsCma import *

driver = RsCma('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 RsCma import *

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

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

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

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

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

driver = RsCma('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'
```

(continues on next page)

(continued from previous page)

```
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

RsCma 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 RsCma 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 RsCma import *

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

(continues on next page)

(continued from previous page)

```
    print(e.args[0])
    print('That took a long time...')

except RsInstrException as e:
    # RsInstrException is a base class for all the RsCma exceptions
    print(e.args[0])
    print('Some other RsCma 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 RsCma, 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 RsCma 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 RsCma has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

RsCma 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 RsCma import *
import time


def my_transfer_handler(args):
```

(continues on next page)

(continued from previous page)

```
    """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 = RsCma('TCPIP::192.168.56.101::INSTR')

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

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

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

import threading
from RsCma import *


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


driver = RsCma('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 RsCma objects with shared session
```

(continues on next page)

(continued from previous page)

```
"""

import threading
from RsCma import *


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

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

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

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

Run the following example:

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

import threading
from RsCma import *


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

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

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

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

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

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

Console output:

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

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

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

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

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

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

driver.utilities.reset()

# Close the session
driver.close()

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

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

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

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

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

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

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

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

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

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

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

from RsCma import *

driver = RsCma('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 AcpOffset

```
# Example value:
value = enums.AcpOffset.LSB
# All values (3x):
LSB | NONE | USB
```

3.2 Activity

```
# Example value:
value = enums.Activity.ACTive
# All values (2x):
ACTive | INACtive
```

3.3 Allow

```
# Example value:
value = enums.Allow.ALLowed
# All values (2x):
ALLowed | NA
```

3.4 AnalogDigital

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

3.5 ArbSamplesRange

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

3.6 ArmedState

```
# Example value:
value = enums.ArmedState.ARMed
# All values (3x):
ARMed | OFF | TRIGgered
```

3.7 AttenuationPort

```
# Example value:
value = enums.AttenuationPort.LOAD
# All values (2x):
LOAD | SOURce
```

3.8 AudioConnector

```
# Example value:
value = enums.AudioConnector.AF1O
# All values (2x):
AF1O | AF2O
```

3.9 AudioSource

```
# Example value:
value = enums.AudioSource.DEM
# All values (6x):
DEM | DEML | DEMR | NONE | UGEN | VOIP
```

3.10 AveragingMode

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

3.11 BandpassFilter

```
# Example value:
value = enums.BandpassFilter.F01M
# All values (5x):
F01M | F05M | F25K | F50K | F8330
```

3.12 Bandwidth

```
# Example value:
value = enums.Bandwidth.FR1K
# All values (5x):
FR1K | FR1K1 | FR1K2 | FR1K3 | FR1K4
```

3.13 BaseScenario

```
# First value:
value = enums.BaseScenario.AUDio
# Last value:
value = enums.BaseScenario.TXTest
# All values (12x):
AUDio | AVIonics | DEXPert | DRXTest | DSPectrum | DTXTest | DXTest | EXPert
RXTest | SEQuencer | SPECtrum | TXTest
```

3.14 BatteryUsage

```
# Example value:
value = enums.BatteryUsage.NAV
# All values (3x):
NAV | REMovable | USED
```

3.15 BerPeriod

```
# Example value:
value = enums.BerPeriod.F36
# All values (2x):
F36 | F48
```

3.16 ByteOrder

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

3.17 CalibType

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

3.18 CatalogFormat

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

3.19 ChannelModeDmr

```
# Example value:
value = enums.ChannelModeDmr.DATA
# All values (2x):
DATA | VOICe
```

3.20 ChannelModeTetra

```
# Example value:
value = enums.ChannelModeTetra.TCH72
# All values (1x):
TCH72
```

3.21 ChannelTypeTetra

```
# Example value:
value = enums.ChannelTypeTetra.CT0
# All values (8x):
CT0 | CT1 | CT2 | CT21 | CT22 | CT24 | CT3 | CT4
```

3.22 CircuitryState

```
# Example value:
value = enums.CircuitryState.ACTive
# All values (2x):
ACTive | PASSive
```

3.23 ClockRate

```
# First value:
value = enums.ClockRate.BPS115200
# Last value:
value = enums.ClockRate.BPS9600
# All values (10x):
BPS115200 | BPS1200 | BPS14400 | BPS19200 | BPS2400 | BPS28800 | BPS38400 | BPS4800
BPS57600 | BPS9600
```

3.24 CrestFactor

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

3.25 DataFormat

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

3.26 DefaultUnitAngle

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

3.27 DefaultUnitCapacity

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

3.28 DefaultUnitCharge

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

3.29 DefaultUnitConductance

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

3.30 DefaultUnitCurrent

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

3.31 DefaultUnitEnergy

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

3.32 DefaultUnitFrequency

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

3.33 DefaultUnitFullScale

```
# Example value:
value = enums.DefaultUnitFullScale.DBFS
# All values (4x):
DBFS | FS | PCT | PPM
```

3.34 DefaultUnitLenght

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

3.35 DefaultUnitPower

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

3.36 DefaultUnitResistor

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

3.37 DefaultUnitTemperature

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

3.38 DefaultUnitTime

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

3.39 DefaultUnitVoltage

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

3.40 DeltaMode

```
# Example value:
value = enums.DeltaMode.MEAS
# All values (3x):
MEAS | NONE | USER
```

3.41 Demodulation

```
# Example value:
value = enums.Demodulation.AM
# All values (6x):
AM | FM | FMSTereo | LSB | PM | USB
```

3.42 DemodulationType

```
# Example value:
value = enums.DemodulationType.FSK4
# All values (1x):
FSK4
```

3.43 Detector

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

3.44 DetectorSimple

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

3.45 DialingMode

```
# Example value:
value = enums.DialingMode.DTMF
# All values (4x):
DTMF | FDIaling | SCAL | SELCall
```

3.46 DigitalSource

```
# Example value:
value = enums.DigitalSource.ARB
# All values (8x):
ARB | DMR | DPMR | NXDN | P25 | POCSag | UDEFined | ZIGBee
```

3.47 DigitalToneMode

```
# Example value:
value = enums.DigitalToneMode.DCS
# All values (6x):
DCS | DTMF | FDIA | NONE | SCAL | SELCall
```

3.48 DigitTimeMode

```
# Example value:
value = enums.DigitTimeMode.EQUal
# All values (2x):
EQUal | INDividual
```

3.49 DirectionIo

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

3.50 DirPwrSensorFwdValue

```
# Example value:
value = enums.DirPwrSensorFwdValue.CCDF
# All values (4x):
CCDF | CFAC | FPWR | PEP
```

3.51 DirPwrSensorRevValue

```
# Example value:
value = enums.DirPwrSensorRevValue.REFL
# All values (4x):
REFL | RLOS | RPWR | SWR
```

3.52 DisplayLanguage

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

3.53 DmrPattern

```
# First value:
value = enums.DmrPattern.BPRB15
# Last value:
value = enums.DmrPattern.SILence
# All values (16x):
BPRB15 | BPRB9 | C153 | O153 | P1031 | PRBS9 | R10A | RA0
RA1 | RFBS | RFMS | RLD | RPRB15 | RPRB9 | RSYR | SILence
```

3.54 DmrPatternB

```
# Example value:
value = enums.DmrPatternB.P1031
# All values (3x):
P1031 | SILence | SYNC
```

3.55 Dstrategy

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

3.56 Eformat

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

3.57 EstartMode

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

3.58 EstopMode

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

3.59 ExpFrequency

```
# Example value:
value = enums.ExpFrequency.CONF
# All values (2x):
CONF | FGEN
```

3.60 ExpressionMode

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

3.61 ExtPwrSensorApp

```
# Example value:
value = enums.ExtPwrSensorApp.EPS
# All values (2x):
EPS | NRTZ
```

3.62 ExtSensorResolution

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

3.63 FftLength

```
# Example value:
value = enums.FftLength.F16K
# All values (3x):
F16K | F4K | F8K
```

3.64 FftOffsetMode

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

3.65 FftSpan

```
# Example value:
value = enums.FftSpan.SP1
# All values (4x):
SP1 | SP10 | SP21 | SP5
```

3.66 FftWindowType

```
# Example value:
value = enums.FftWindowType.BLHA
# All values (5x):
BLHA | FLTP | HAMMing | HANN | RECTangle
```

3.67 FileSave

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

3.68 FilterDigital

```
# Example value:
value = enums.FilterDigital.COSine
# All values (4x):
COSine | GAUSs | RRC | SINC
```

3.69 FilterNxDn

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

3.70 FilterType

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

3.71 FreqCounterMode

```
# Example value:
value = enums.FreqCounterMode.HW
# All values (2x):
HW | SW
```

3.72 FskMode

```
# Example value:
value = enums.FskMode.FSK2
# All values (2x):
FSK2 | FSK4
```

3.73 GeneratorCoupling

```
# Example value:
value = enums.GeneratorCoupling.GEN1
# All values (5x):
GEN1 | GEN2 | GEN3 | GEN4 | OFF
```

3.74 GeneratorCouplingVoIp

```
# Example value:
value = enums.GeneratorCouplingVoIp.GEN3
# All values (3x):
GEN3 | GEN4 | OFF
```

3.75 GeneratorFilter

```
# Example value:
value = enums.GeneratorFilter.COS
# All values (5x):
COS | GAUSs | RC | RRC | SINC
```

3.76 GeneratorState

```
# Example value:
value = enums.GeneratorState.ADJusted
# All values (7x):
ADJusted | AUTonomous | COUPled | INValid | OFF | ON | PENDing
```

3.77 HighpassFilter

```
# Example value:
value = enums.HighpassFilter.F300
# All values (2x):
F300 | OFF
```

3.78 HighpassFilterExtended

```
# Example value:
value = enums.HighpassFilterExtended.F300
# All values (4x):
F300 | F50 | F6 | OFF
```

3.79 IlsLetter

```
# Example value:
value = enums.IlsLetter.X
# All values (2x):
X | Y
```

3.80 IlsTab

```
# Example value:
value = enums.IlsTab.GSLope
# All values (2x):
GSLope | LOCalizer
```

3.81 Impedance

```
# Example value:
value = enums.Impedance.IHOL
# All values (5x):
IHOL | R150 | R300 | R50 | R600
```

3.82 ImpulseLength

```
# Example value:
value = enums.ImpulseLength.T
# All values (5x):
T | T2 | T4 | T6 | T8
```

3.83 InputConnector

```
# Example value:
value = enums.InputConnector.RFCom
# All values (2x):
RFCom | RFIN
```

3.84 InterfererMode

```
# Example value:
value = enums.InterfererMode.AM
# All values (5x):
AM | CW | FM | NONE | PM
```

3.85 IqFormat

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

3.86 LeftRightDirection

```
# Example value:
value = enums.LeftRightDirection.LEFT
# All values (2x):
LEFT | RIGHt
```

3.87 LevelEditMode

```
# Example value:
value = enums.LevelEditMode.INDividual
# All values (2x):
INDividual | TOTal
```

3.88 LinkDirectionDmr

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

3.89 LinkDirectionTetra

```
# Example value:
value = enums.LinkDirectionTetra.DLNK
# All values (2x):
DLNK | ULNK
```

3.90 LockRangeExternal

```
# Example value:
value = enums.LockRangeExternal.INV
# All values (4x):
INV | MEDium | NARRow | WIDE
```

3.91 LockRangeInternal

```
# Example value:
value = enums.LockRangeInternal.INV
# All values (3x):
INV | MEDium | NARRow
```

3.92 LowHigh

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

3.93 LowpassFilter

```
# Example value:
value = enums.LowpassFilter.F15K
# All values (4x):
F15K | F3K | F4K | OFF
```

3.94 LowpassFilterExtended

```
# Example value:
value = enums.LowpassFilterExtended.F15K
# All values (6x):
F15K | F255 | F3K | F3K4 | F4K | OFF
```

3.95 LteChannelBandwidth

```
# Example value:
value = enums.LteChannelBandwidth.F10M
# All values (4x):
F10M | F20M | F3M | F5M
```

3.96 MagnitudeUnit

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

3.97 MarkerFunction

```
# Example value:
value = enums.MarkerFunction.MAX
# All values (6x):
MAX | MAXL | MAXN | MAXR | MAXV | MIN
```

3.98 MarkerPlacement

```
# Example value:
value = enums.MarkerPlacement.ABSolute
# All values (2x):
ABSolute | RELative
```

3.99 MeasAccuracy

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

3.100 MeasState

```
# First value:
value = enums.MeasState.ACTive
# Last value:
value = enums.MeasState.RUN
# All values (10x):
ACTive | ADJusted | ALIVe | FROZen | INValid | OFF | PENDing | QUEued
RDY | RUN
```

3.101 ModeTetra

```
# Example value:
value = enums.ModeTetra.DQPS45
# All values (1x):
DQPS45
```

3.102 ModulationScheme

```
# Example value:
value = enums.ModulationScheme.AM
# All values (8x):
AM | ARB | CW | FM | FMSTereo | LSB | PM | USB
```

3.103 NotchPath

```
# Example value:
value = enums.NotchPath.AF
# All values (3x):
AF | SPDif | VOIP
```

3.104 NrtDevice

```
# Example value:
value = enums.NrtDevice.N14
# All values (3x):
N14 | N43 | N44
```

3.105 NxdnPattern

```
# First value:
value = enums.NxdnPattern.P1011
# Last value:
value = enums.NxdnPattern.SILence
# All values (14x):
P1011 | P1031 | PRBS15 | PRBS9 | R10A | RA0 | RA1 | RAW1
RAW2 | RLD | RPRB15 | RPRB9 | RSYR | SILence
```

3.106 NxdnPatternB

```
# First value:
value = enums.NxdnPatternB.CUST
# Last value:
value = enums.NxdnPatternB.STD1
# All values (13x):
CUST | P1031 | PRBS9 | R1031 | R10A | RA0 | RA1 | RLD
RPRB15 | RPRB9 | RSYR | SILence | STD1
```

3.107 OperationMode

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

3.108 OptionsProductType

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

3.109 OptionsScope

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

3.110 OptionValidity

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

3.111 OscillatorType

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

3.112 OutputConnector

```
# Example value:
value = enums.OutputConnector.RFCom
# All values (2x):
RFCom | RFOut
```

3.113 OverviewType

```
# Example value:
value = enums.OverviewType.FFT
# All values (3x):
FFT | NONE | OSCilloscope
```

3.114 P25Mode

```
# Example value:
value = enums.P25Mode.C4FM
# All values (2x):
C4FM | CQPSk
```

3.115 P25Pattern

```
# First value:
value = enums.P25Pattern.BUSY
# Last value:
value = enums.P25Pattern.SILence
# All values (15x):
BUSY | C4FM | CALibration | IDLE | INTerference | P1011 | R10A | RA0
RA1 | RAW1 | RLD | RPRB15 | RPRB9 | RSYR | SILence
```

3.116 PagerType

```
# Example value:
value = enums.PagerType.ALPHanumeric
# All values (3x):
ALPHanumeric | NUMeric | TONLy
```

3.117 PathCoupling

```
# Example value:
value = enums.PathCoupling.AC
# All values (2x):
AC | DC
```

3.118 PatternTetra

```
# Example value:
value = enums.PatternTetra.S1
# All values (3x):
S1 | S2 | S3
```

3.119 PayloadType

```
# Example value:
value = enums.PayloadType.P1011
# All values (2x):
P1011 | SILence
```

3.120 PayloadTypeTetra

```
# Example value:
value = enums.PayloadTypeTetra.ALLO
# All values (5x):
ALLO | ALLZero | ALTE | PRBS9 | USER
```

3.121 PowerMode

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

3.122 PowerSignalDirection

```
# Example value:
value = enums.PowerSignalDirection.AUTO
# All values (3x):
AUTO | FWD | REV
```

3.123 PreDeEmphasis

```
# Example value:
value = enums.PreDeEmphasis.OFF
# All values (4x):
OFF | T50 | T75 | T750
```

3.124 ProtocolMode

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

3.125 PtFiveFilter

```
# Example value:
value = enums.PtFiveFilter.C4FM
# All values (2x):
C4FM | RC
```

3.126 PulseShapingFilter

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

3.127 PulseShapingUserFilter

```
# Example value:
value = enums.PulseShapingUserFilter.COS
# All values (5x):
COS | GAUSs | NXRX | RRC | SINC
```

3.128 PwrFilterType

```
# Example value:
value = enums.PwrFilterType.NARRow
# All values (3x):
NARRow | UDEF | WIDE
```

3.129 RbwFilterType

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

3.130 RefFreqSource

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

3.131 Relative

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

3.132 Repeat

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

3.133 RepeatMode

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

3.134 ResourceState

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

3.135 ResultStatus

```
# First value:
value = enums.ResultStatus.DC
# Last value:
value = enums.ResultStatus.ULEU
# All values (11x):
DC | INV | NAV | NCAP | OFF | OFL | OK | ON
UFL | ULEL | ULEU
```

3.136 ScreenshotFormat

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

3.137 SearchExtent

```
# Example value:
value = enums.SearchExtent.FULL
# All values (3x):
FULL | OFFLevel | ONLevel
```

3.138 SearchRoutine

```
# Example value:
value = enums.SearchRoutine.ADELay
# All values (6x):
ADELay | RIFBandwidth | RSENsitivity | RSQuelch | SSNR | TSENsitivity
```

3.139 SearchRoutinePath

```
# Example value:
value = enums.SearchRoutinePath.AFI1
# All values (3x):
AFI1 | AFI2 | VOIP
```

3.140 SelCallStandard

```
# Example value:
value = enums.SelCallStandard.CCIR
# All values (8x):
CCIR | DZVei | EEA | EIA | PZVei | ZVEI1 | ZVEI2 | ZVEI3
```

3.141 SelCalStandard

```
# Example value:
value = enums.SelCalStandard.SCAL16
# All values (3x):
SCAL16 | SCAL32 | UDEFind
```

3.142 SharingModeTetra

```
# Example value:
value = enums.SharingModeTetra.CARRier
# All values (4x):
CARRier | CONTinuous | MMCH | TRAFfic
```

3.143 SignalSlope

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

3.144 SignalSlopeExt

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

3.145 SignalSource

```
# First value:
value = enums.SignalSource.AFI1
# Last value:
value = enums.SignalSource.SPIR
# All values (14x):
AFI1 | AFI2 | AFIB | FCHL | FCHR | FILE | GEN1 | GEN2
GEN3 | GEN4 | GENB | SPIL | SPIN | SPIR
```

3.146 SingDualTonesType

```
# Example value:
value = enums.SingDualTonesType.DTONes
# All values (2x):
DTONes | STONes
```

3.147 SingDualToneType

```
# Example value:
value = enums.SingDualToneType.DTONe
# All values (2x):
DTONe | STONe
```

3.148 SipState

```
# Example value:
value = enums.SipState.ERRor
# All values (3x):
ERRor | ESTablished | TERMinated
```

3.149 SlowDecay

```
# Example value:
value = enums.SlowDecay.OFF
# All values (5x):
OFF | X10 | X2 | X3 | X4
```

3.150 SoundSource

```
# Example value:
value = enums.SoundSource.AFONe
# All values (7x):
AFONe | AVIO | DEModulator | GENone | GENThree | LAN | SPDif
```

3.151 SpanMode

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

3.152 SpecAnApp

```
# Example value:
value = enums.SpecAnApp.FREQ
# All values (2x):
FREQ | ZERO
```

3.153 Standard

```
# Example value:
value = enums.Standard.CUSTom
# All values (8x):
CUSTom | DMR | DPMR | LTE | NXDN | P25 | SPECtrum | TETRa
```

3.154 StandardB

```
# Example value:
value = enums.StandardB.CUSTom
# All values (7x):
CUSTom | DMR | DPMR | LTE | NXDN | P25 | TETRa
```

3.155 StandardDigital

```
# Example value:
value = enums.StandardDigital.DMR
# All values (3x):
DMR | PTFive | TETRa
```

3.156 Statistic

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

3.157 StatRegFormat

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

3.158 Status

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

3.159 StopCondition

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

3.160 SubTab

```
# First value:
value = enums.SubTab.AFResults
# Last value:
value = enums.SubTab.TRIM
# All values (9x):
AFResults | FFT | FMSTereo | MULTitone | OSC | OVERview | RFResults | TONes
TRIM
```

3.161 SubTabAudioMeas

```
# Example value:
value = enums.SubTabAudioMeas.AFResults
# All values (5x):
AFResults | FFT | OSC | OVERview | TRIM
```

3.162 SubTabDigitalMeas

```
# Example value:
value = enums.SubTabDigitalMeas.BER
# All values (3x):
BER | OVERview | SINFo
```

3.163 SubTabRoutines

```
# Example value:
value = enums.SubTabRoutines.CHARt
# All values (2x):
CHARt | TABLe
```

3.164 SubTabVseMeas

```
# First value:
value = enums.SubTabVseMeas.CONStellation
# Last value:
value = enums.SubTabVseMeas.SYMResults
# All values (11x):
CONStellation | DEMod | EYEDiagram | LTE | NXDNresults | OVERview | PVTResults |␣
↪RFResults
SPECtrum | SYMDistr | SYMResults
```

3.165 SupplyMode

```
# Example value:
value = enums.SupplyMode.BATTery
# All values (2x):
BATTery | MAINs
```

3.166 TabSplit

```
# Example value:
value = enums.TabSplit.SPLit
# All values (2x):
SPLit | TAB
```

3.167 TargetParameter

```
# Example value:
value = enums.TargetParameter.NPEK
# All values (5x):
NPEK | PNPA | PPEK | RMS | RMSQ
```

3.168 TargetParType

```
# Example value:
value = enums.TargetParType.SINad
# All values (4x):
SINad | SNDNratio | SNNRatio | SNRatio
```

3.169 TestModeTetra

```
# Example value:
value = enums.TestModeTetra.SIDecoding
# All values (3x):
SIDecoding | T1 | VSE
```

3.170 TestPlanState

```
# First value:
value = enums.TestPlanState.EDITmode
# Last value:
value = enums.TestPlanState.SERRor
# All values (10x):
EDITmode | FINished | IDLE | LOADing | NOAVailable | NOLoaded | OPTMissing | PAUSed
RUNNing | SERRor
```

3.171 TimeoutMode

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

3.172 ToneMode

```
# Example value:
value = enums.ToneMode.NOISe
# All values (4x):
NOISe | NONE | SQUare | STONe
```

3.173 ToneTypeA

```
# First value:
value = enums.ToneTypeA.DTMF
# Last value:
value = enums.ToneTypeA.STONe
# All values (9x):
DTMF | DTONe | FDIaling | MTONe | NOISe | SCAL | SELCall | SQUare
STONe
```

3.174 ToneTypeB

```
# Example value:
value = enums.ToneTypeB.CTCSs
# All values (4x):
CTCSs | DCS | NONE | SUBTone
```

3.175 TraceB

```
# Example value:
value = enums.TraceB.AVERage
# All values (5x):
AVERage | CURRent | MAXimum | MINimum | TDOMmain
```

3.176 TraceC

```
# Example value:
value = enums.TraceC.AVERage
# All values (3x):
AVERage | CURRent | MAXimum
```

3.177 Transmission

```
# Example value:
value = enums.Transmission.EFR9600
# All values (3x):
EFR9600 | EHR4800 | EHR9600
```

3.178 TriggerCouplingAin

```
# Example value:
value = enums.TriggerCouplingAin.DEMod
# All values (4x):
DEMod | NONE | SIN | VOIP
```

3.179 TriggerCouplingDemod

```
# Example value:
value = enums.TriggerCouplingDemod.AIN
# All values (4x):
AIN | NONE | SIN | VOIP
```

3.180 TriggerCouplingDigital

```
# Example value:
value = enums.TriggerCouplingDigital.AIN
# All values (3x):
AIN | DEMod | NONE
```

3.181 TriggerMode

```
# Example value:
value = enums.TriggerMode.AUTO
# All values (4x):
AUTO | FRUN | NORMal | SINGle
```

3.182 TriggerSourceAf

```
# Example value:
value = enums.TriggerSourceAf.AF1
# All values (2x):
AF1 | AF2
```

3.183 TriggerSourceDemod

```
# Example value:
value = enums.TriggerSourceDemod.DEMod
# All values (3x):
DEMod | LEFT | RIGHt
```

3.184 TtlInterface

```
# Example value:
value = enums.TtlInterface.WIRE1
# All values (2x):
WIRE1 | WIRE2
```

3.185 TxAfSource

```
# Example value:
value = enums.TxAfSource.AF1O
# All values (3x):
AF1O | AF2O | VOIP
```

3.186 UpDownDirection

```
# Example value:
value = enums.UpDownDirection.DOWN
# All values (2x):
DOWN | UP
```

3.187 UserDefPattern

```
# Example value:
value = enums.UserDefPattern.PRBS6
# All values (2x):
PRBS6 | PRBS9
```

3.188 UserRole

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

3.189 VoIpCodec

```
# Example value:
value = enums.VoIpCodec.ALAW
# All values (2x):
ALAW | ULAW
```

3.190 VoIpSource

```
# Example value:
value = enums.VoIpSource.AFI1
# All values (4x):
AFI1 | AFI2 | GEN3 | GEN4
```

3.191 VphaseDirection

```
# Example value:
value = enums.VphaseDirection.FROM
# All values (2x):
FROM | TO
```

3.192 WeightingFilter

```
# Example value:
value = enums.WeightingFilter.AWEighting
# All values (4x):
AWEighting | CCITt | CMESsage | OFF
```

3.193 Xdivision

```
# First value:
value = enums.Xdivision.M1
# Last value:
value = enums.Xdivision.U500
# All values (19x):
M1 | M10 | M100 | M2 | M20 | M200 | M5 | M50
M500 | S1 | U1 | U10 | U100 | U2 | U20 | U200
U5 | U50 | U500
```

3.194 XrtInputConnector

```
# Example value:
value = enums.XrtInputConnector.RF1
# All values (8x):
RF1 | RF2 | RF3 | RF4 | RF5 | RF6 | RF7 | RF8
```

3.195 YesNoStatus

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

3.196 ZigBeeMode

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

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 AudioInput

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

4.3 AudioOutput

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

4.4 Battery

```
# First value:
value = repcap.Battery.Ix1
# Values (2x):
Ix1 | Ix2
```

4.5 Bit

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

4.6 Channel

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

4.7 Connector

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

4.8 FrequencyLobe

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

4.9 Instrument

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

4.10 InternalGen

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

4.11 Marker

```
# First value:
value = repcap.Marker.Nr1
# Range:
Nr1 .. Nr5
# All values (5x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5
```

4.12 MarkerOther

```
# First value:
value = repcap.MarkerOther.Nr2
# Values (4x):
Nr2 | Nr3 | Nr4 | Nr5
```

4.13 Notch

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

4.14 Relay

```
# First value:
value = repcap.Relay.Ix1
# Values (2x):
Ix1 | Ix2
```

4.15 ToneNumber

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

4.16 TTL

```
# First value:
value = repcap.TTL.Ix1
# Values (2x):
Ix1 | Ix2
```

4.17 Window

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

CHAPTER

FIVE

EXAMPLES

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

```
# Example of using R&S CMA180 auto-generated instrument driver module RsCma for Python 3

from RsCma import *  # install from pypi.org

RsCma.assert_minimum_version('1.5.70')
cma = RsCma('TCPIP::10.112.1.116::INSTR', True, False)
print(f'CMA Identification: {cma.utilities.idn_string}')

# SYSTem:DISPlay:UPDate
cma.system.display.set_update(False)

# SOURce:AFRF:GEN:STATe
cma.source.afRf.generator.state.set(True)

# SOURce:AFRF:GEN:RFSettings:FREQuency
cma.source.afRf.generator.rfSettings.set_frequency(100E3)

# SOURce:AFRF:GEN:RFSettings:LEVel
cma.source.afRf.generator.rfSettings.set_level(-20.0)

# CONFigure:GPRF:MEAS:SPECtrum:FREQuency:SPAN:MODE
cma.configure.gprfMeasurement.spectrum.frequency.span.set_mode(enums.SpanMode.FSWeep)

# CONFigure:GPRF:MEAS:SPECtrum:SCOunt
cma.configure.gprfMeasurement.spectrum.set_scount(10)

# CONFigure:GPRF:MEAS:SPECtrum:REPetition
cma.configure.gprfMeasurement.spectrum.set_repetition(enums.Repeat.SINGleshot)

# CONFigure:GPRF:MEAS:SPECtrum:FREQuency:SPAN
cma.configure.gprfMeasurement.spectrum.frequency.span.set_value(1.1E6)

# CONFigure:GPRF:MEAS:SPECtrum:FREQuency:CENTer
cma.configure.gprfMeasurement.spectrum.frequency.set_center(100E3)

# INITiate:GPRF:MEAS:SPECtrum
cma.gprfMeasurement.spectrum.initiate()
cma.gprfMeasurement.spectrum.initiate_with_opc()
```

(continues on next page)

(continued from previous page)

```
# READ:GPRF:MEAS:POWer:CURRent?
results_power = cma.gprfMeasurement.power.current.read()

# FETCh:GPRF:MEAS:SPECtrum:RMS:CURRent?
results_spectrum = cma.gprfMeasurement.spectrum.rms.current.fetch()

# SOURce:AFRF:GENerator:IGENerator<nr>:MTONe:TONE<no>:ENABle
# option A: stating all the repcaps in the method call
# sends: SOURce:AFRF:GENerator:IGENerator2:MTONe:TONE4:ENABle ON
cma.source.afRf.generator.internalGenerator.multiTone.tone.enable.set(True, repcap.
↪InternalGen.Nr2, repcap.ToneNumber.Nr4)

# option B - setting default values of the repcaps in the groups:
cma.source.afRf.generator.internalGenerator.repcap_internalGen_set(repcap.InternalGen.
↪Nr2)
cma.source.afRf.generator.internalGenerator.multiTone.tone.repcap_toneNumber_set(repcap.
↪ToneNumber.Nr4)

# Then use the method call without defining the repcaps - they stay at the default value
# sends: SOURce:AFRF:GENerator:IGENerator2:MTONe:TONE4:ENABle ON
cma.source.afRf.generator.internalGenerator.multiTone.tone.enable.set(True)

# cloning instances:
igen3 = cma.source.afRf.generator.internalGenerator.clone()
igen3.repcap_internalGen_set(repcap.InternalGen.Nr3)
igen3.multiTone.tone.enable.set(True)
# sends: SOURce:AFRF:GENerator:IGENerator3:MTONe:TONE4:ENABle ON

igen3.multiTone.tone.repcap_toneNumber_set(repcap.ToneNumber.Nr7)
# sends: SOURce:AFRF:GENerator:IGENerator3:MTONe:TONE7:ENABle ON

cma.close()
```

CHAPTER

SIX

RSCMA API STRUCTURE

Global RepCaps

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

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

2178 total commands, 21 Subgroups, 0 group commands

Initializes new RsCma session.

Parameter options tokens examples:

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

- `ViClearExeMode = Disabled` - viClear() execution mode. Default: `execute_on_all`
- `OpcQueryAfterWrite = True` - same as driver.utilities.opc_query_after_write = True. Default: `False`
- `StbInErrorCheck = False` - if true, the driver checks errors with *STB? If false, it uses SYST:ERR?. Default: `True`
- `LoggingMode = On` - Sets the logging status right from the start. Default: `Off`
- `LoggingName = 'MyDevice'` - Sets the name to represent the session in the log entries. Default: `'resource_name'`
- `LogToGlobalTarget = True` - Sets the logging target to the class-property previously set with RsCma.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 RsCma session.

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

Creates a new RsCma 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 AfRf

class AfRfCls

AfRf commands group definition. 669 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1 Measurement

class MeasurementCls

Measurement commands group definition. 669 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.1 Digital

SCPI Command:

```
INITiate:AFRF:MEASurement<Instance>:DIGital
STOP:AFRF:MEASurement<Instance>:DIGital
ABORt:AFRF:MEASurement<Instance>:DIGital
```

class DigitalCls

Digital commands group definition. 51 total commands, 5 Subgroups, 3 group commands

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

```
# SCPI: ABORt:AFRF:MEASurement<Instance>:DIGital
driver.afRf.measurement.digital.abort()
```

Stops the measurement.

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

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

```
# SCPI: INITiate:AFRF:MEASurement<Instance>:DIGital
driver.afRf.measurement.digital.initiate()
```

Starts or continues the measurement.

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

stop() → None

```
# SCPI: STOP:AFRF:MEASurement<Instance>:DIGital
driver.afRf.measurement.digital.stop()
```

Pauses the measurement.

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

```
# SCPI: STOP:AFRF:MEASurement<Instance>:DIGital
driver.afRf.measurement.digital.stop_with_opc()
```

Pauses the measurement.

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

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

Cloning the Group

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

Subgroups

6.1.1.1.1 Dmr

class DmrCls

Dmr commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.1.1.1 BitErrorRate

class BitErrorRateCls

BitErrorRate 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.afRf.measurement.digital.dmr.bitErrorRate.clone()
```

Subgroups

6.1.1.1.1.2 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:CURRent
CALCulate:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:CURRent
```

class CurrentCls

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

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:CURRent
value: enums.ResultStatus = driver.afRf.measurement.digital.dmr.bitErrorRate.
↪current.calculate()
```

Queries BER measurement results for the DMR standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:CURRent
value: float = driver.afRf.measurement.digital.dmr.bitErrorRate.current.fetch()
```

Queries BER measurement results for the DMR standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:CURRent
value: float = driver.afRf.measurement.digital.dmr.bitErrorRate.current.read()
```

Queries BER measurement results for the DMR standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

6.1.1.1.1.3 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:MAXimum
CALCulate:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:MAXimum
READ:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:MAXimum
```

class MaximumCls

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

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:MAXimum
value: enums.ResultStatus = driver.afRf.measurement.digital.dmr.bitErrorRate.
↪maximum.calculate()
```

Queries BER measurement results for the DMR standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:MAXimum
value: float = driver.afRf.measurement.digital.dmr.bitErrorRate.maximum.fetch()
```

Queries BER measurement results for the DMR standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:DMR:BERate:MAXimum
value: float = driver.afRf.measurement.digital.dmr.bitErrorRate.maximum.read()
```

Queries BER measurement results for the DMR standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

6.1.1.1.1.4 PoOff

class PoOffCls

PoOff 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.afRf.measurement.digital.dmr.poOff.clone()
```

Subgroups

6.1.1.1.1.5 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:POOFf:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:DMR:POOFf:CURRent
```

class CurrentCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power_Slot_0: float: Unit: dBm
- Power_Slot_1: float: Unit: dBm

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:POOFf:CURRent
value: ResultData = driver.afRf.measurement.digital.dmr.poOff.current.fetch()
```

Queries the current power level of the on-time slot 'Power ON (Slot 0) ' and the current power level on the off-time slot 'Power OFF (Slot 1) '.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:DMR:POOFf:CURRent
value: ResultData = driver.afRf.measurement.digital.dmr.poOff.current.read()
```

Queries the current power level of the on-time slot 'Power ON (Slot 0) ' and the current power level on the off-time slot 'Power OFF (Slot 1) '.

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

6.1.1.1.1.6 Power

class PowerCls

Power 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.afRf.measurement.digital.dmr.power.clone()
```

Subgroups

6.1.1.1.1.7 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:POWer:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:DMR:POWer:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:POWer:CURRent
value: float = driver.afRf.measurement.digital.dmr.power.current.fetch()
```

No command help available

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

return
power: No help available

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:DMR:POWer:CURRent
value: float = driver.afRf.measurement.digital.dmr.power.current.read()
```

No command help available

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

return
power: No help available

6.1.1.1.1.8 Sinfo

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:SINFo
READ:AFRF:MEASurement<Instance>:DIGital:DMR:SINFo
```

class SinfoCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Color_Code: int: Color code as part of the PDU content
- Source_Address: int: Source address as part of the PDU content
- Target_Address: int: Target address as part of the PDU content
- Pi: int: No parameter help available
- Pflag: int: Protect flag as part of the PDU content
- Flco: int: No parameter help available
- Fid: int: Feature set ID as part of the PDU content
- Data_Type: int: Data type as part of the PDU content
- Broadcast: int: Broadcast operation as part of the service options
- Privacy: int: Privacy operation as part of the service options
- Pl: int: Priority level as part of the service options
- Emergency: int: Emergency operation as part of the service options
- Ovcm: int: Open voice call mode as part of the service options Range: 0 to 15

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:DMR:SINFo
value: ResultData = driver.afRf.measurement.digital.dmr.sinfo.fetch()
```

Queries signal information parameters for the DMR standard. SIgnal information includes the PDU content and service options.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:DMR:SINFo
value: ResultData = driver.afRf.measurement.digital.dmr.sinfo.read()
```

Queries signal information parameters for the DMR standard. SIgnal information includes the PDU content and service options.

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

6.1.1.1.2 PtFive

class PtFiveCls

PtFive commands group definition. 10 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.1.2.1 BitErrorRate

class BitErrorRateCls

BitErrorRate 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.afRf.measurement.digital.ptFive.bitErrorRate.clone()
```

Subgroups

6.1.1.1.2.2 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:CURRent
CALCulate:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:CURRent
```

class CurrentCls

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

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:CURRent
value: enums.ResultStatus = driver.afRf.measurement.digital.ptFive.bitErrorRate.
↪current.calculate()
```

Queries BER measurement results for the P25 standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:CURRent
value: float = driver.afRf.measurement.digital.ptFive.bitErrorRate.current.
↪fetch()
```

Queries BER measurement results for the P25 standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:CURRent
value: float = driver.afRf.measurement.digital.ptFive.bitErrorRate.current.
↪read()
```

Queries BER measurement results for the P25 standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

6.1.1.1.2.3 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:MAXimum
CALCulate:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:MAXimum
READ:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:MAXimum
```

class MaximumCls

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

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:MAXimum
value: enums.ResultStatus = driver.afRf.measurement.digital.ptFive.bitErrorRate.
↪maximum.calculate()
```

Queries BER measurement results for the P25 standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:MAXimum
value: float = driver.afRf.measurement.digital.ptFive.bitErrorRate.maximum.
↪fetch()
```

Queries BER measurement results for the P25 standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:PTFive:BERate:MAXimum
value: float = driver.afRf.measurement.digital.ptFive.bitErrorRate.maximum.
↪read()
```

Queries BER measurement results for the P25 standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

6.1.1.1.2.4 Power

class PowerCls

Power 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.afRf.measurement.digital.ptFive.power.clone()
```

Subgroups

6.1.1.1.2.5 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:POWer:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:PTFive:POWer:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:POWer:CURRent
value: float = driver.afRf.measurement.digital.ptFive.power.current.fetch()
```

Queries the RMS power level of the RF input signal for standard ‘P25’.

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

return
power: Unit: dBm

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:PTFive:POWer:CURRent
value: float = driver.afRf.measurement.digital.ptFive.power.current.read()
```

Queries the RMS power level of the RF input signal for standard ‘P25’.

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

return
power: Unit: dBm

6.1.1.1.2.6 Sinfo

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:SINFo
READ:AFRF:MEASurement<Instance>:DIGital:PTFive:SINFo
```

class SinfoCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See ‘Reliability indicator values’
- Network_Access_Code: str: Network access code in hexadecimal representation of the network identifier
- Link_Control_Format: int: Link control format of the link control word
- Mf_Id: int: Manufacture ID of the link control word
- Emergency: int: Emergency field of the link control word
- Reserved: int: Reserved field of the link control word
- Target_Id: str: Target ID in hexadecimal representation of the link control word
- Source_Id: str: Source ID in hexadecimal representation of the link control word
- Message_Indicator: int: Message indicator of the encryption sync word
- Alg_Id: int: Algorithm ID of the encryption sync word
- Key_Id: int: Key ID of the encryption sync word

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:PTFive:SINFo
value: ResultData = driver.afRf.measurement.digital.ptFive.sinfo.fetch()
```

Queries signal information parameters for the P25 standard. Signal information includes the network identifier, the link control word and the encryption sync word.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:PTFive:SINFo
value: ResultData = driver.afRf.measurement.digital.ptFive.sinfo.read()
```

Queries signal information parameters for the P25 standard. Signal information includes the network identifier, the link control word and the encryption sync word.

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

6.1.1.1.3 State

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:STATe
```

class StateCls

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

fetch() → ResourceState

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:STATe
value: enums.ResourceState = driver.afRf.measurement.digital.state.fetch()
```

Queries the main measurement state.

return
meas_state: OFF | RUN | RDY | PENDing | ADJusted | QUEued | ACTive | INValid

Cloning the Group

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

Subgroups

6.1.1.1.3.1 All

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:STATe:ALL
```

class AllCls

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

fetch() → List[ResourceState]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:STATe:ALL
value: List[enums.ResourceState] = driver.afRf.measurement.digital.state.all.
↪fetch()
```

Queries the main measurement state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: OFF | RUN | RDY | PENDing | ADJusted | QUEued | ACTive | INValid

6.1.1.1.4 Tetra

class TetraCls

Tetra commands group definition. 13 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.1.4.1 BitErrorRate

class BitErrorRateCls

BitErrorRate 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.afRf.measurement.digital.tetra.bitErrorRate.clone()
```

Subgroups

6.1.1.1.4.2 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:CURRent
CALCulate:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:CURRent
```

class CurrentCls

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

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:CURRent
value: enums.ResultStatus = driver.afRf.measurement.digital.tetra.bitErrorRate.
↪current.calculate()
```

Queries BER measurement results for the TETRA standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:CURRent
value: float = driver.afRf.measurement.digital.tetra.bitErrorRate.current.
↪fetch()
```

Queries BER measurement results for the TETRA standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:CURRent
value: float = driver.afRf.measurement.digital.tetra.bitErrorRate.current.read()
```

Queries BER measurement results for the TETRA standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

6.1.1.1.4.3 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:MAXimum
CALCulate:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:MAXimum
READ:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:MAXimum
```

class MaximumCls

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

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:MAXimum
value: enums.ResultStatus = driver.afRf.measurement.digital.tetra.bitErrorRate.
↪maximum.calculate()
```

Queries BER measurement results for the TETRA standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:MAXimum
value: float = driver.afRf.measurement.digital.tetra.bitErrorRate.maximum.
↪fetch()
```

Queries BER measurement results for the TETRA standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TETRa:BERate:MAXimum
value: float = driver.afRf.measurement.digital.tetra.bitErrorRate.maximum.read()
```

Queries BER measurement results for the TETRA standard. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Unit: %

6.1.1.1.4.4 FreqError

class FreqErrorCls

FreqError 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.afRf.measurement.digital.tetra.freqError.clone()
```

Subgroups

6.1.1.1.4.5 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:FERRor:CURRent
CALCulate:AFRF:MEASurement<Instance>:DIGital:TETRa:FERRor:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:TETRa:FERRor:CURRent
```

class CurrentCls

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

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:TETRa:FERRor:CURRent
value: enums.ResultStatus = driver.afRf.measurement.digital.tetra.freqError.
↪current.calculate()
```

No command help available

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

return
freq_error: No help available

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:FERRor:CURRent
value: float = driver.afRf.measurement.digital.tetra.freqError.current.fetch()
```

No command help available

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

return
freq_error: No help available

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TETRa:FERRor:CURRent
value: float = driver.afRf.measurement.digital.tetra.freqError.current.read()
```

No command help available

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

return
freq_error: No help available

6.1.1.1.4.6 Power

class PowerCls

Power 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.afRf.measurement.digital.tetra.power.clone()
```

Subgroups

6.1.1.1.4.7 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:POWer:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:TETRa:POWer:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:POWer:CURRent
value: float = driver.afRf.measurement.digital.tetra.power.current.fetch()
```

No command help available

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

return
power: No help available

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TETRa:POWer:CURRent
value: float = driver.afRf.measurement.digital.tetra.power.current.read()
```

No command help available

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

return
power: No help available

6.1.1.1.4.8 Sinfo

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:SINFo
READ:AFRF:MEASurement<Instance>:DIGital:TETRa:SINFo
```

class SinfoCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- System_Code: int: System code
- Sharing_Mode: enums.SharingModeTetra: CONTinuous | CARRier | MMCH | TRAFfic Sharing mode
- Reserved_Frame: int: Reserved frame
- Dtx: enums.Allow: NA | ALLowed
- Frame_Ext: enums.Allow: NA | ALLowed
- Broadcast: enums.Allow: NA | ALLowed
- Cell_Service_Level: enums.Allow: NA | ALLowed
- Late_Entry: enums.Allow: NA | ALLowed
- Bcc: int: No parameter help available
- Mcc: int: No parameter help available
- Mnc: int: No parameter help available
- Time_Slot_Num: int: No parameter help available
- Frame_Num: int: No parameter help available
- Multiframe_Num: int: No parameter help available

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TETRa:SINFo
value: ResultData = driver.afRf.measurement.digital.tetra.sinfo.fetch()
```

Queries signal information parameters for the TETRA standard when operating in downlink/forward direction: CONF:AFRF:MEAS:DIG:TETR:LDIR DLNK Signal information includes the system code, test modes, frame and service information.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TETRa:SINFo
value: ResultData = driver.afRf.measurement.digital.tetra.sinfo.read()
```

Queries signal information parameters for the TETRA standard when operating in downlink/forward direction: CONF:AFRF:MEAS:DIG:TETR:LDIR DLNK Signal information includes the system code, test modes, frame and service information.

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

6.1.1.1.5 Ttl

class TtlCls

Ttl commands group definition. 11 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.1.5.1 BitErrorRate

class BitErrorRateCls

BitErrorRate commands group definition. 11 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.digital.ttl.bitErrorRate.clone()
```

Subgroups

6.1.1.1.5.2 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:AVERage
CALCulate:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:AVERage
READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:AVERage
```

class AverageCls

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

calculate() → float

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:AVERage
value: float or bool = driver.afRf.measurement.digital.ttl.bitErrorRate.average.
↪calculate()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: (float or boolean) Range: 0 % to 100 %, Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:AVERage
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.average.fetch()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:AVERage
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.average.read()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

6.1.1.1.5.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:CURRent
CALCulate:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:CURRent
READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:CURRent
```

class CurrentCls

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

calculate() → float

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:CURRent
value: float or bool = driver.afRf.measurement.digital.ttl.bitErrorRate.current.
↪calculate()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: (float or boolean) Range: 0 % to 100 %, Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:CURRent
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.current.fetch()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:CURRent
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.current.read()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

6.1.1.1.5.4 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:MAXimum
CALCulate:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:MAXimum
READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:MAXimum
```

class MaximumCls

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

calculate() → float

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:MAXimum
value: float or bool = driver.afRf.measurement.digital.ttl.bitErrorRate.maximum.
↪calculate()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: (float or boolean) Range: 0 % to 100 %, Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:MAXimum
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.maximum.fetch()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:MAXimum
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.maximum.read()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

6.1.1.1.5.5 StandardDev

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:SDEViation
READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:SDEViation
```

class StandardDevCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:SDEViation
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.standardDev.
↪fetch()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:DIGital:TTL:BERate:SDEViation
value: float = driver.afRf.measurement.digital.ttl.bitErrorRate.standardDev.
↪read()
```

Queries BER measurement results for the TTL path. CALCulate commands return error indicators instead of measurement values.

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

return
ber: Range: 0 % to 100 %, Unit: %

6.1.1.2 Frequency

class FrequencyCls

Frequency commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.2.1 Counter

SCPI Command:

```
INITiate:AFRF:MEASurement<Instance>:FREQuency:COUNter
ABORt:AFRF:MEASurement<Instance>:FREQuency:COUNter
FETCh:AFRF:MEASurement<Instance>:FREQuency:COUNter
```

class CounterCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of signal peak Range: 0 Hz to 3 GHz, Unit: Hz
- Level: float: Power of signal peak Unit: dBm

abort() → None

```
# SCPI: ABORt:AFRF:MEASurement<Instance>:FREQuency:COUNter
driver.afRf.measurement.frequency.counter.abort()
```

Aborts the search procedure for an RF signal. The configured RF settings are not modified.

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

```
# SCPI: ABORt:AFRF:MEASurement<Instance>:FREQuency:COUNter
driver.afRf.measurement.frequency.counter.abort_with_opc()
```

Aborts the search procedure for an RF signal. The configured RF settings are not modified.

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

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

fetch() → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:FREQuency:COUNter
value: FetchStruct = driver.afRf.measurement.frequency.counter.fetch()
```

Queries the search procedure results.

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

initiate() → None

```
# SCPI: INITiate:AFRF:MEASurement<Instance>:FREQuency:COUNter
driver.afRf.measurement.frequency.counter.initiate()
```

Starts the search procedure to find an RF signal.

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

```
# SCPI: INITiate:AFRF:MEASurement<Instance>:FREQuency:COUNter
driver.afRf.measurement.frequency.counter.initiate_with_opc()
```

Starts the search procedure to find an RF signal.

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

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

Cloning the Group

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

Subgroups

6.1.1.2.1.1 FreqError

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:FREQuency:COUNter:FERRor
```

class FreqErrorCls

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

fetch() → int

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:FREQuency:COUNter:FERRor
value: int = driver.afRf.measurement.frequency.counter.freqError.fetch()
```

Queries the frequency error determined by the search procedure. The error is calculated as counted frequency minus configured analyzer frequency.

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

return
frequency_error: Range: -3 GHz to 3 GHz, Unit: Hz

6.1.1.3 MultiEval

SCPI Command:

```
INITiate:AFRF:MEASurement<Instance>:MEValuation
STOP:AFRF:MEASurement<Instance>:MEValuation
ABORt:AFRF:MEASurement<Instance>:MEValuation
```

class MultiEvalCls

MultiEval commands group definition. 532 total commands, 13 Subgroups, 3 group commands

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

```
# SCPI: ABORt:AFRF:MEASurement<Instance>:MEValuation
driver.afRf.measurement.multiEval.abort()
```

Stops the analyzer.

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

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

```
# SCPI: INITiate:AFRF:MEASurement<Instance>:MEValuation
driver.afRf.measurement.multiEval.initiate()
```

Starts or continues the analyzer.

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

stop() → None

```
# SCPI: STOP:AFRF:MEASurement<Instance>:MEValuation
driver.afRf.measurement.multiEval.stop()
```

Pauses the analyzer.

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

```
# SCPI: STOP:AFRF:MEASurement<Instance>:MEValuation
driver.afRf.measurement.multiEval.stop_with_opc()
```

Pauses the analyzer.

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

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

Cloning the Group

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

Subgroups

6.1.1.3.1 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.afRf.measurement.multiEval.audioInput.repcap_audioInput_get()
driver.afRf.measurement.multiEval.audioInput.repcap_audioInput_set(repcap.AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 48 total commands, 4 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

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

Subgroups

6.1.1.3.1.1 AfSignal

class AfSignalCls

AfSignal 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.afRf.measurement.multiEval.audioInput.afSignal.clone()
```

Subgroups

6.1.1.3.1.2 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:AVERage
```

class AverageCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪average.fetch(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪average.read(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

6.1.1.3.1.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:CURRent
```

class CurrentCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪current.fetch(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪current.read(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

6.1.1.3.1.4 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:DEViation
```

class DeviationCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz

- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪deviation.fetch(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪deviation.read(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

6.1.1.3.1.5 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:MAXimum
```

class MaximumCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪maximum.fetch(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.audioInput.afSignal.
↪maximum.read(audioInput = repcap.AudioInput.Default)
```

Query the AF frequency and level results measured for an AF input path.

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

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

6.1.1.3.1.6 First

class FirstCls

First commands group definition. 16 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.audioInput.first.clone()
```

Subgroups

6.1.1.3.1.7 AfSignal

class AfSignalCls

AfSignal 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.afRf.measurement.multiEval.audioInput.first.afSignal.clone()
```

Subgroups

6.1.1.3.1.8 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:AVERage
```

class AverageCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪average.fetch()
```

Query the AF frequency and level results measured for an AF input path.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪average.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.9 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:CURRent
```

class CurrentCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪current.fetch()
```

Query the AF frequency and level results measured for an AF input path.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪current.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.10 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:DEViation
```

class DeviationCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪deviation.fetch()
```

Query the AF frequency and level results measured for an AF input path.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪deviation.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.11 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:MAXimum
```

class MaximumCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪maximum.fetch()
```

Query the AF frequency and level results measured for an AF input path.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.audioInput.first.afSignal.
↪maximum.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.12 Level

class LevelCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.audioInput.first.level.clone()
```

Subgroups

6.1.1.3.1.13 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.audioInput.first.level.delta.clone()
```

Subgroups

6.1.1.3.1.14 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪average.fetch()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪average.read()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

6.1.1.3.1.15 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪current.fetch()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪current.read()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

6.1.1.3.1.16 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪deviation.fetch()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪deviation.read()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

6.1.1.3.1.17 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:FIRSt:LEVel:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪maximum.fetch()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:FIRSt:LEVel:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.audioInput.first.level.delta.
↪maximum.read()
```

Query delta results for AF level at AF1.

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

return
level: Unit: V

6.1.1.3.1.18 Frequency

class FrequencyCls

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

Cloning the Group

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

Subgroups

6.1.1.3.1.19 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.audioInput.frequency.delta.clone()
```

Subgroups

6.1.1.3.1.20 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:AVERage
```

class AverageCls

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

fetch(*audioInput=AudioInput.Default*) → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪average.fetch(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

read(*audioInput=AudioInput.Default*) → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪average.read(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

6.1.1.3.1.21 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:CURRent
```

class CurrentCls

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

fetch(*audioInput=AudioInput.Default*) → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪current.fetch(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

read(*audioInput=AudioInput.Default*) → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪current.read(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

6.1.1.3.1.22 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:DEViation
```

`class DeviationCls`

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

`fetch`(*audioInput=AudioInput.Default*) → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪deviation.fetch(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

`read`(*audioInput=AudioInput.Default*) → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪deviation.read(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

6.1.1.3.1.23 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>:FREQuency:DELTa:MAXimum
```

class MaximumCls

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

fetch(*audioInput=AudioInput.Default*) → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪maximum.fetch(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

read(*audioInput=AudioInput.Default*) → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN<nr>
↪:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.audioInput.frequency.delta.
↪maximum.read(audioInput = repcap.AudioInput.Default)
```

Query delta results for AF frequency for AF path.

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

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

return
frequency: Unit: Hz

6.1.1.3.1.24 Second

class SecondCls

Second commands group definition. 16 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.audioInput.second.clone()
```

Subgroups

6.1.1.3.1.25 AfSignal

class AfSignalCls

AfSignal 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.afRf.measurement.multiEval.audioInput.second.afSignal.clone()
```

Subgroups

6.1.1.3.1.26 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:AVERage
```

class AverageCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.average.fetch()
```

Query the AF frequency and level results measured for an AF input path.

return

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.average.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.27 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:CURRent
```

class CurrentCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.current.fetch()
```

Query the AF frequency and level results measured for an AF input path.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.current.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.28 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:DEViation
```

class DeviationCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.deviation.fetch()
```

Query the AF frequency and level results measured for an AF input path.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.deviation.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.29 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:MAXimum
```

class MaximumCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level: float: Effective level of the AC component of the measured AF signal Unit: V
- Dc_Level: float: Level of the DC component of the measured AF signal (input coupling DC required) Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.maximum.fetch()
```

Query the AF frequency and level results measured for an AF input path.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.audioInput.second.
↪afSignal.maximum.read()
```

Query the AF frequency and level results measured for an AF input path.

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

6.1.1.3.1.30 Level

class LevelCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.audioInput.second.level.clone()
```

Subgroups

6.1.1.3.1.31 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.audioInput.second.level.delta.clone()
```

Subgroups

6.1.1.3.1.32 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪average.fetch()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪average.read()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

6.1.1.3.1.33 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪current.fetch()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪current.read()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

6.1.1.3.1.34 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪deviation.fetch()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪deviation.read()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

6.1.1.3.1.35 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:AIN:SECond:LEVel:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪maximum.fetch()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:AIN:SECond:LEVel:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.audioInput.second.level.delta.
↪maximum.read()
```

Query delta results for AF level at AF2.

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

return
level: Unit: V

6.1.1.3.2 DemodLeft

class DemodLeftCls

DemodLeft commands group definition. 16 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.2.1 AfSignal

class AfSignalCls

AfSignal commands group definition. 16 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodLeft.afSignal.clone()
```

Subgroups

6.1.1.3.2.2 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:AVERage
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.average.
↪fetch()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:AVERage
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.average.
↪read()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.2.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:CURRent
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.current.
↪fetch()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:CURRent
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.current.
↪read()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.2.4 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.demodLeft.afSignal.delta.clone()
```

Subgroups

6.1.1.3.2.5 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪average.fetch()
```

Query delta results of the AF frequency for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪average.read()
```

Query delta results of the AF frequency for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

6.1.1.3.2.6 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪current.fetch()
```

Query delta results for demodulation frequencies of an FM stereo signal.

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

return
frequency: No help available

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪current.read()
```

Query delta results for demodulation frequencies of an FM stereo signal.

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

return
frequency: No help available

6.1.1.3.2.7 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪deviation.fetch()
```

Query delta results of the AF frequency for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪deviation.read()
```

Query delta results of the AF frequency for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

6.1.1.3.2.8 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪maximum.fetch()
```

Query delta results of the AF frequency for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMLeft:AFSignal:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.delta.
↪maximum.read()
```

Query delta results of the AF frequency for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

6.1.1.3.2.9 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DEViation
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.deviation.
↪fetch()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:DEViation
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.deviation.
↪read()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.2.10 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:MAXimum
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.maximum.
↪fetch()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMLeft:AFSignal:MAXimum
value: float = driver.afRf.measurement.multiEval.demodLeft.afSignal.maximum.
↪read()
```

Query the AF frequency results for the left demodulator channel. For FM stereo, these results are related to the left audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.3 DemodRight

class DemodRightCls

DemodRight commands group definition. 16 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.3.1 AfSignal

class AfSignalCls

AfSignal commands group definition. 16 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodRight.afSignal.clone()
```

Subgroups

6.1.1.3.3.2 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:AVERage
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.average.
↪fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:AVERage
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.average.
↪read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.3.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:CURRent
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.current.
↪fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:CURRent
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.current.
↪read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.3.4 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.demodRight.afSignal.delta.clone()
```

Subgroups

6.1.1.3.3.5 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪average.fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪average.read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

6.1.1.3.3.6 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪current.fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪current.read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

6.1.1.3.3.7 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪deviation.fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪deviation.read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

6.1.1.3.3.8 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪maximum.fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEMRight:AFSignal:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.delta.
↪maximum.read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Delta frequency value of the AF signal Range: 0 Hz to 21 kHz, Unit: Hz

6.1.1.3.3.9 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DEViation
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.deviation.
↪fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:DEViation
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.deviation.
↪read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.3.10 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:MAXimum
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.maximum.
↪fetch()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEMRight:AFSignal:MAXimum
value: float = driver.afRf.measurement.multiEval.demodRight.afSignal.maximum.
↪read()
```

Query the AF frequency results for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel.

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

return
frequency: Frequency of the AF signal Unit: Hz

6.1.1.3.4 Demodulation

class DemodulationCls

Demodulation commands group definition. 76 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.4.1 Fdeviation

class FdeviationCls

Fdeviation commands group definition. 28 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodulation.fdeviation.clone()
```

Subgroups

6.1.1.3.4.2 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: Hz
- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float or bool: Positive peak value Unit: Hz
- Mpeak: float or bool: Negative peak value Unit: Hz
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: Hz
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float: Positive peak value Unit: Hz
- Mpeak: float: Negative peak value Unit: Hz
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fdeviation.average.calculate()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪average.fetch()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪average.read()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.3 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: Hz
- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float or bool: Positive peak value Unit: Hz
- Mpeak: float or bool: Negative peak value Unit: Hz
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: Hz

- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float: Positive peak value Unit: Hz
- Mpeak: float: Negative peak value Unit: Hz
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fdeviation.current.calculate()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪current.fetch()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪current.read()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.4 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:DEViation
```

class DeviationCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: Hz
- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float or bool: Positive peak value Unit: Hz
- Mpeak: float or bool: Negative peak value Unit: Hz
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: Hz
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float: Positive peak value Unit: Hz
- Mpeak: float: Negative peak value Unit: Hz
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fdeviation.deviation.calculate()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪deviation.fetch()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪deviation.read()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.5 Maximum

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:MAXimum
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:MAXimum
```

class MaximumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: Hz
- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float or bool: Positive peak value Unit: Hz
- Mpeak: float or bool: Negative peak value Unit: Hz
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: Hz
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: Hz
- Ppeak: float: Positive peak value Unit: Hz
- Mpeak: float: Negative peak value Unit: Hz

- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:MAXimum
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fdeviation.maximum.calculate()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪maximum.fetch()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fdeviation.
↪maximum.read()
```

Queries the demodulation results for FM or FM stereo demodulation. A statistical evaluation of the frequency deviation or multiplex deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.6 Peak

class PeakCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.clone()
```

Subgroups

6.1.1.3.4.7 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.demodulation.fdeviation.peak.delta.clone()
```

Subgroups

6.1.1.3.4.8 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.average.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.average.read()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

6.1.1.3.4.9 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.current.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.current.read()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

6.1.1.3.4.10 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.deviation.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.deviation.read()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

6.1.1.3.4.11 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:PEAK:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.maximum.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:PEAK:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.peak.
↪delta.maximum.read()
```

Query the demodulation results of frequency deviation for delta measurement. The peak values of a mono signal are delivered.

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

return
peak: Unit: Hz

6.1.1.3.4.12 Rms

class RmsCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.clone()
```

Subgroups

6.1.1.3.4.13 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.demodulation.fdeviation.rms.delta.clone()
```

Subgroups

6.1.1.3.4.14 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.average.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.average.read()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

6.1.1.3.4.15 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.current.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.current.read()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

6.1.1.3.4.16 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.deviation.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.deviation.read()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

6.1.1.3.4.17 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FDEViation:RMS:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.maximum.fetch()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FDEViation:RMS:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.fdeviation.rms.
↪delta.maximum.read()
```

Query the demodulation results of frequency deviation for delta measurement. The RMS values of a mono signal are delivered.

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

return
rms: Unit: Hz

6.1.1.3.4.18 FmStereo

class FmStereoCls

FmStereo commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodulation.fmStereo.clone()
```

Subgroups

6.1.1.3.4.19 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See ‘Reliability indicator values’
- Audio_Dev_Left: float or bool: Peak frequency deviation due to the left audio channel Unit: Hz
- Audio_Dev_Right: float or bool: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float or bool: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float or bool: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float or bool: Peak frequency deviation due to the signal in the RDS band Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Audio_Dev_Left: float: Peak frequency deviation due to the left audio channel Unit: Hz
- Audio_Dev_Right: float: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float: Peak frequency deviation due to the signal in the RDS band Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fmStereo.average.calculate()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪average.fetch()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪average.read()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.20 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Audio_Dev_Left: float or bool: Peak frequency deviation due to the left audio channel Unit: Hz
- Audio_Dev_Right: float or bool: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float or bool: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float or bool: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float or bool: Peak frequency deviation due to the signal in the RDS band Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Audio_Dev_Left: float: Peak frequency deviation due to the left audio channel Unit: Hz
- Audio_Dev_Right: float: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float: Peak frequency deviation due to the signal in the RDS band Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fmStereo.current.calculate()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪current.fetch()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪current.read()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.21 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:DEViation
```

class DeviationCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Audio_Dev_Left: float or bool: Peak frequency deviation due to the left audio channel Unit: Hz
- Audio_Dev_Right: float or bool: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float or bool: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float or bool: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float or bool: Peak frequency deviation due to the signal in the RDS band Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Audio_Dev_Left: float: Peak frequency deviation due to the left audio channel Unit: Hz
- Audio_Dev_Right: float: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float: Peak frequency deviation due to the signal in the RDS band Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fmStereo.deviation.calculate()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪deviation.fetch()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪deviation.read()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.22 Maximum

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:MAXimum
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FMSTereo:MAXimum
```

class MaximumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Audio_Dev_Left: float or bool: Peak frequency deviation due to the left audio channel Unit: Hz

- Audio_Dev_Right: float or bool: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float or bool: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float or bool: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float or bool: Peak frequency deviation due to the signal in the RDS band Unit: Hz

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Audio_Dev_Left: float: Peak frequency deviation due to the left audio channel Unit: Hz
- Audio_Dev_Right: float: Peak frequency deviation due to the right audio channel Unit: Hz
- Pilot_Deviation: float: Peak frequency deviation due to the pilot tone Unit: Hz
- Pilot_Freq_Error: float: Frequency error of the pilot tone Unit: Hz
- Rds_Deviation: float: Peak frequency deviation due to the signal in the RDS band Unit: Hz

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:MAXimum
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪fmStereo.maximum.calculate()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪maximum.fetch()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FMSTereo:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.fmStereo.
↪maximum.read()
```

Query the demodulation results for the individual components of an FM stereo signal. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.23 Frequency

class FrequencyCls

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

Cloning the Group

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

Subgroups

6.1.1.3.4.24 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.demodulation.frequency.delta.clone()
```

Subgroups

6.1.1.3.4.25 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪average.fetch()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪average.read()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

6.1.1.3.4.26 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪current.fetch()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪current.read()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

6.1.1.3.4.27 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪deviation.fetch()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪deviation.read()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

6.1.1.3.4.28 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:FREQuency:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪maximum.fetch()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.frequency.delta.
↪maximum.read()
```

Query delta results for demodulation frequency of mono signal.

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

return
frequency: Unit: Hz

6.1.1.3.4.29 ModDepth

class ModDepthCls

ModDepth commands group definition. 16 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodulation.modDepth.clone()
```

Subgroups

6.1.1.3.4.30 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:AVERage
```

class AverageCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: %
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: %
- Ppeak: float: Positive peak value Unit: %
- Mpeak: float: Negative peak value Unit: %

- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪average.fetch()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪average.read()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

6.1.1.3.4.31 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:CURRent
```

class CurrentCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: %
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: %
- Ppeak: float: Positive peak value Unit: %
- Mpeak: float: Negative peak value Unit: %
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪current.fetch()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪current.read()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

6.1.1.3.4.32 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.demodulation.modDepth.delta.clone()
```

Subgroups

6.1.1.3.4.33 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪average.fetch()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪average.read()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

6.1.1.3.4.34 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪current.fetch()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪current.read()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

6.1.1.3.4.35 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪deviation.fetch()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪deviation.read()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

6.1.1.3.4.36 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪maximum.fetch()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.demodulation.modDepth.delta.
↪maximum.read()
```

Query the demodulation delta results for AM demodulation.

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

return
mod_depth: Range: 0.01 dB to 100.00 dB , Unit: dB

6.1.1.3.4.37 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:DEViation
```

class DeviationCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: %
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: %
- Ppeak: float: Positive peak value Unit: %
- Mpeak: float: Negative peak value Unit: %
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪deviation.fetch()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:MDEPth:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪deviation.read()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

6.1.1.3.4.38 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:MAXimum
```

class MaximumCls

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: %
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: %
- Ppeak: float: Positive peak value Unit: %
- Mpeak: float: Negative peak value Unit: %
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪maximum.fetch()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:MDEPth:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.modDepth.
↪maximum.read()
```

Query the demodulation results for AM demodulation. A statistical evaluation of the modulation depth is returned.

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

6.1.1.3.4.39 Pdeviation

class PdeviationCls

Pdeviation commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.demodulation.pdeviation.clone()
```

Subgroups

6.1.1.3.4.40 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: rad
- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float or bool: Positive peak value Unit: rad
- Mpeak: float or bool: Negative peak value Unit: rad
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: rad

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: rad
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float: Positive peak value Unit: rad
- Mpeak: float: Negative peak value Unit: rad
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: rad

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪pdeviation.average.calculate()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪average.fetch()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:AVERage
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪average.read()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.41 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: rad

- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float or bool: Positive peak value Unit: rad
- Mpeak: float or bool: Negative peak value Unit: rad
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: rad

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: rad
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float: Positive peak value Unit: rad
- Mpeak: float: Negative peak value Unit: rad
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: rad

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪pdeviation.current.calculate()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪current.fetch()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:CURRent
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪current.read()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.42 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:DEViation
```

class DeviationCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: rad
- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float or bool: Positive peak value Unit: rad
- Mpeak: float or bool: Negative peak value Unit: rad
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: rad

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: rad
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float: Positive peak value Unit: rad
- Mpeak: float: Negative peak value Unit: rad
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: rad

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪pdeviation.deviation.calculate()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪deviation.fetch()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:DEViation
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪deviation.read()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.4.43 Maximum

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:MAXimum
FETCh:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:DEModulation:PDEViation:MAXimum
```

class MaximumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float or bool: RMS average value Unit: rad
- Rms_Sqrt_2: float or bool: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float or bool: Positive peak value Unit: rad
- Mpeak: float or bool: Negative peak value Unit: rad
- Mp_Peak_Average: enums.ResultStatus: Peak-to-peak value divided by 2 Unit: rad

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: RMS average value Unit: rad
- Rms_Sqrt_2: float: RMS result multiplied with the square root of 2 Unit: rad
- Ppeak: float: Positive peak value Unit: rad
- Mpeak: float: Negative peak value Unit: rad
- Mp_Peak_Average: float: Peak-to-peak value divided by 2 Unit: rad

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:MAXimum
value: CalculateStruct = driver.afRf.measurement.multiEval.demodulation.
↪pdeviation.maximum.calculate()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪maximum.fetch()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:DEModulation:PDEViation:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.demodulation.pdeviation.
↪maximum.read()
```

Query the demodulation results for PM demodulation. A statistical evaluation of the phase deviation is returned. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.5 Fft

class FftCls

Fft commands group definition. 92 total commands, 8 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.5.1 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.afRf.measurement.multiEval.fft.audioInput.repcap_audioInput_get()
driver.afRf.measurement.multiEval.fft.audioInput.repcap_audioInput_set(repcap.AudioInput.
↪Nr1)
```

class AudioInputCls

AudioInput commands group definition. 11 total commands, 2 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.audioInput.clone()
```

Subgroups

6.1.1.3.5.2 Marker

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<nr>:MARKer<mnr>
```

class MarkerCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Absolute_Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *freq_value: float*, *audioInput=AudioInput.Default*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<nr>:MARKer<mnr>
value: FetchStruct = driver.afRf.measurement.multiEval.fft.audioInput.marker.
↪fetch(trace = enums.Statistic.AVERage, freq_value = 1.0, audioInput = repcap.
↪AudioInput.Default, marker = repcap.Marker.Nr1)
```

Move marker number <mnr> to a specified x-value and return the absolute coordinates. Absolute placement is used. Select the trace to be evaluated and the x-value.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param freq_value
X-value for which the coordinates are queried Range: 0 Hz to 21 kHz

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

param marker
optional repeated capability selector. Default value: Nr1

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.audioInput.marker.clone()
```

Subgroups

6.1.1.3.5.3 Absolute

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<nr>:MARKer<mnr>:ABSolute
```

class AbsoluteCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *audioInput=AudioInput.Default*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<nr>:MARKer<mnr>
↪:ABSolute
value: FetchStruct = driver.afRf.measurement.multiEval.fft.audioInput.marker.
↪absolute.fetch(trace = enums.Statistic.AVERage, function = enums.
↪MarkerFunction.MAX, audioInput = repcap.AudioInput.Default, marker = repcap.
↪Marker.Nr1)
```

Query the absolute coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the

query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

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

param marker
optional repeated capability selector. Default value: Nr1

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

6.1.1.3.5.4 Relative

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<nr>:MARKer<mnr>:RELative
```

`class RelativeCls`

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

`class FetchStruct`

Response structure. Fields:

- Dx_Value: float: Delta X value of the marker relative to the reference marker Unit: Hz
- Dy_Value: float: Delta Y value of the marker relative to the reference marker Unit: Depends on input path and demodulation type

`fetch`(*trace: Statistic*, *function: MarkerFunction = None*, *audioInput=AudioInput.Default*, *markerOther=MarkerOther.Nr2*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<nr>:MARKer<mnr>
↪:RELative
value: FetchStruct = driver.afRf.measurement.multiEval.fft.audioInput.marker.
↪relative.fetch(trace = enums.Statistic.AVERage, function = enums.
↪MarkerFunction.MAX, audioInput = repcap.AudioInput.Default, markerOther =␣
↪repcap.MarkerOther.Nr2)
```

Query the relative coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

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

param markerOther
optional repeated capability selector. Default value: Nr2

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

6.1.1.3.5.5 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.afRf.measurement.multiEval.fft.audioInput.power.clone()
```

Subgroups

6.1.1.3.5.6 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:AVERage
```

class AverageCls

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

fetch(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪average.fetch(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBV

read(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪average.read(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBV

6.1.1.3.5.7 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:CURRent
```

class CurrentCls

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

fetch(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪current.fetch(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBV

read(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪current.read(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right) Unit: dBV

6.1.1.3.5.8 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MAXimum
```

class MaximumCls

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

fetch(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪maximum.fetch(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right) Unit: dBV

read(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪maximum.read(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right) Unit: dBV

6.1.1.3.5.9 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MINimum
```

class MinimumCls

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

fetch(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪minimum.fetch(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBV

read(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:AIN<Nr>:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.audioInput.power.
↪minimum.read(audioInput = repcap.AudioInput.Default)
```

Query the contents of the spectrum diagram for an AF input path.

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

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBV

6.1.1.3.5.10 DemodLeft

class DemodLeftCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.demodLeft.clone()
```

Subgroups

6.1.1.3.5.11 Fdeviation

class FdeviationCls

Fdeviation 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.afRf.measurement.multiEval.fft.demodLeft.fdeviation.clone()
```

Subgroups

6.1.1.3.5.12 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪average.fetch()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪average.read()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.13 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪current.fetch()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪current.read()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.14 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪maximum.fetch()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪maximum.read()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.15 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMLeft:FDEViation:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪minimum.fetch()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMLeft:FDEViation:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodLeft.fdeviation.
↪minimum.read()
```

Query the contents of the spectrum diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, these results are related to the left audio channel.

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

return
power: Comma-separated list of 1793 frequency deviation or audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.16 DemodRight

class DemodRightCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.demodRight.clone()
```

Subgroups

6.1.1.3.5.17 Fdeviation

class FdeviationCls

Fdeviation 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.afRf.measurement.multiEval.fft.demodRight.fdeviation.clone()
```

Subgroups

6.1.1.3.5.18 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.average.fetch()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.average.read()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.19 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.current.fetch()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.current.read()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.20 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.maximum.fetch()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.maximum.read()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.21 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEMRight:FDEViation:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.minimum.fetch()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEMRight:FDEViation:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodRight.
↪fdeviation.minimum.read()
```

Query the contents of the spectrum diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
power: Comma-separated list of 1793 audio deviation values (diagram from left to right) Unit: dBHz

6.1.1.3.5.22 Demodulation<Channel>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.afRf.measurement.multiEval.fft.demodulation.repcap_channel_get()
driver.afRf.measurement.multiEval.fft.demodulation.repcap_channel_set(repcap.Channel.Nr1)
```

class DemodulationCls

Demodulation commands group definition. 35 total commands, 5 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.demodulation.clone()
```

Subgroups

6.1.1.3.5.23 LsbPower

class LsbPowerCls

LsbPower 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.afRf.measurement.multiEval.fft.demodulation.lsbPower.clone()
```

Subgroups

6.1.1.3.5.24 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.average.fetch()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.average.read()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.25 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.current.fetch()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.current.read()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.26 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.maximum.fetch()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.maximum.read()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.27 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:LSBPower:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.minimum.fetch()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:LSBPower:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪lsbPower.minimum.read()
```

Query the contents of the spectrum diagram for SSB-LSB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.28 Marker

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation<nr>:MARKer<mnr>
```

class MarkerCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Absolute_Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *freq_value: float = None*, *channel=Channel.Default*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation<nr>:MARKer
↪<mnr>
value: FetchStruct = driver.afRf.measurement.multiEval.fft.demodulation.marker.
↪fetch(trace = enums.Statistic.AVERage, freq_value = 1.0, channel = repcap.
↪Channel.Default, marker = repcap.Marker.Nr1)
```

Move marker number <mnr> to a specified x-value and return the absolute coordinates. Absolute placement is used. Select the trace to be evaluated and the x-value.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param freq_value
X-value for which the coordinates are queried Range: 0 Hz to 21 kHz

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

param marker
optional repeated capability selector. Default value: Nr1

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.demodulation.marker.clone()
```

Subgroups

6.1.1.3.5.29 Absolute

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation<nr>:MARKer<mnr>:ABSolute
```

class AbsoluteCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *channel=Channel.Default*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation<nr>:MARKer
↪<mnr>:ABSolute
value: FetchStruct = driver.afRf.measurement.multiEval.fft.demodulation.marker.
↪absolute.fetch(trace = enums.Statistic.AVERage, function = enums.
↪MarkerFunction.MAX, channel = repcap.Channel.Default, marker = repcap.Marker.
↪Nr1)
```

Query the absolute coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

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

param marker
optional repeated capability selector. Default value: Nr1

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

6.1.1.3.5.30 Relative

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation<nr>:MARKer<mnr>:RELative
```

class RelativeCls

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

class FetchStruct

Response structure. Fields:

- Dx_Value: float: Delta X value of the marker relative to the reference marker Unit: Hz
- Dy_Value: float: Delta Y value of the marker relative to the reference marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *channel=Channel.Default*, *markerOther=MarkerOther.Nr2*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation<nr>:MARKer
↪<mnr>:RELative
value: FetchStruct = driver.afRf.measurement.multiEval.fft.demodulation.marker.
↪relative.fetch(trace = enums.Statistic.AVERage, function = enums.
↪MarkerFunction.MAX, channel = repcap.Channel.Default, markerOther = repcap.
↪MarkerOther.Nr2)
```

Query the relative coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

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

param markerOther
optional repeated capability selector. Default value: Nr2

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

6.1.1.3.5.31 ModDepth

class ModDepthCls

ModDepth 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.afRf.measurement.multiEval.fft.demodulation.modDepth.clone()
```

Subgroups

6.1.1.3.5.32 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.average.fetch()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.average.read()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

6.1.1.3.5.33 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.current.fetch()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.current.read()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

6.1.1.3.5.34 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.maximum.fetch()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.maximum.read()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

6.1.1.3.5.35 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:MDEPth:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.minimum.fetch()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:MDEPth:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪modDepth.minimum.read()
```

Query the contents of the spectrum diagram for AM demodulation.

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

return
power: Comma-separated list of 1793 modulation depth values (diagram from left to right) Unit: dB

6.1.1.3.5.36 Pdeviation

class PdeviationCls

Pdeviation 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.afRf.measurement.multiEval.fft.demodulation.pdeviation.clone()
```

Subgroups

6.1.1.3.5.37 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.average.fetch()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.average.read()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

6.1.1.3.5.38 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.current.fetch()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.current.read()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

6.1.1.3.5.39 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.maximum.fetch()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.maximum.read()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

6.1.1.3.5.40 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:PDEViation:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.minimum.fetch()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:PDEViation:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪pdeviation.minimum.read()
```

Query the contents of the spectrum diagram for PM demodulation.

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

return
power: Comma-separated list of 1793 phase-deviation values (diagram from left to right) Unit: dBrad

6.1.1.3.5.41 UsbPower

class UsbPowerCls

UsbPower 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.afRf.measurement.multiEval.fft.demodulation.usbPower.clone()
```

Subgroups

6.1.1.3.5.42 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.average.fetch()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.average.read()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.43 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.current.fetch()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.current.read()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.44 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.maximum.fetch()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.maximum.read()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.45 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:DEModulation:USBPower:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.minimum.fetch()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:FFT:DEModulation:USBPower:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.demodulation.
↪usbPower.minimum.read()
```

Query the contents of the spectrum diagram for SSB-USB demodulation.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: dBFS

6.1.1.3.5.46 Spdif<Channel>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.afRf.measurement.multiEval.fft.spdif.repcap_channel_get()
driver.afRf.measurement.multiEval.fft.spdif.repcap_channel_set(repcap.Channel.Nr1)
```

class SpdifCls

Spdif commands group definition. 3 total commands, 1 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.spdif.clone()
```

Subgroups

6.1.1.3.5.47 Marker

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SIN<nr>:MARKer<mnr>
```

class MarkerCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Absolute_Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *freq_value: float*, *channel=Channel.Default*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SIN<nr>:MARKer<mnr>
value: FetchStruct = driver.afRf.measurement.multiEval.fft.spdif.marker.
↪fetch(trace = enums.Statistic.AVERage, freq_value = 1.0, channel = repcap.
↪Channel.Default, marker = repcap.Marker.Nr1)
```

Move marker number <mnr> to a specified x-value and return the absolute coordinates. Absolute placement is used. Select the trace to be evaluated and the x-value.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param freq_value
X-value for which the coordinates are queried Range: 0 Hz to 21 kHz

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

param marker
optional repeated capability selector. Default value: Nr1

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.spdif.marker.clone()
```

Subgroups

6.1.1.3.5.48 Absolute

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SIN<nr>:MARKer<mnr>:ABSolute
```

class AbsoluteCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *channel=Channel.Default*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SIN<nr>:MARKer<mnr>
↪:ABSolute
value: FetchStruct = driver.afRf.measurement.multiEval.fft.spdif.marker.
↪absolute.fetch(trace = enums.Statistic.AVERage, function = enums.
↪MarkerFunction.MAX, channel = repcap.Channel.Default, marker = repcap.Marker.
↪Nr1)
```

Query the absolute coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

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

param marker
optional repeated capability selector. Default value: Nr1

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

6.1.1.3.5.49 Relative

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SIN<nr>:MARKer<mnr>:RELative
```

class RelativeCls

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

class FetchStruct

Response structure. Fields:

- Dx_Value: float: Delta X value of the marker relative to the reference marker Unit: Hz
- Dy_Value: float: Delta Y value of the marker relative to the reference marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *channel=Channel.Default*, *markerOther=MarkerOther.Nr2*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SIN<nr>:MARKer<mnr>
↪:RELative
value: FetchStruct = driver.afRf.measurement.multiEval.fft.spdif.marker.
↪relative.fetch(trace = enums.Statistic.AVERage, function = enums.
↪MarkerFunction.MAX, channel = repcap.Channel.Default, markerOther = repcap.
↪MarkerOther.Nr2)
```

Query the relative coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

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

param markerOther
optional repeated capability selector. Default value: Nr2

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

6.1.1.3.5.50 SpdifLeft

class SpdifLeftCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.spdifLeft.clone()
```

Subgroups

6.1.1.3.5.51 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.afRf.measurement.multiEval.fft.spdifLeft.power.clone()
```

Subgroups

6.1.1.3.5.52 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪average.fetch()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪average.read()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.53 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪current.fetch()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪current.read()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.54 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪maximum.fetch()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪maximum.read()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.55 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪minimum.fetch()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINLeft:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifLeft.power.
↪minimum.read()
```

Query the contents of the spectrum diagram for the left SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.56 SpdifRight

class SpdifRightCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.spdifRight.clone()
```

Subgroups

6.1.1.3.5.57 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.afRf.measurement.multiEval.fft.spdifRight.power.clone()
```

Subgroups

6.1.1.3.5.58 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪average.fetch()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪average.read()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.59 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪current.fetch()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪current.read()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.60 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪maximum.fetch()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪maximum.read()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.61 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪minimum.fetch()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:SINRight:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.spdifRight.power.
↪minimum.read()
```

Query the contents of the spectrum diagram for the right SPDIF channel.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.62 Voip

class VoipCls

Voip commands group definition. 11 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.voip.clone()
```

Subgroups

6.1.1.3.5.63 Marker

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:MARKer<mnr>
```

class MarkerCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Absolute_Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *freq_value: float*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:MARKer<mnr>
value: FetchStruct = driver.afRf.measurement.multiEval.fft.voip.marker.
↪fetch(trace = enums.Statistic.AVERage, freq_value = 1.0, marker = repcap.
↪Marker.Nr1)
```

Move marker number <mnr> to a specified x-value and return the absolute coordinates. Absolute placement is used. Select the trace to be evaluated and the x-value.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param freq_value
X-value for which the coordinates are queried Range: 0 Hz to 21 kHz

param marker
optional repeated capability selector. Default value: Nr1

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.fft.voip.marker.clone()
```

Subgroups

6.1.1.3.5.64 Absolute

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:MARKer<mnr>:ABSolute
```

class AbsoluteCls

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

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Yvalue: float: Y-value of the marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:MARKer<mnr>
↪:ABSolute
value: FetchStruct = driver.afRf.measurement.multiEval.fft.voip.marker.absolute.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪marker = repcap.Marker.Nr1)
```

Query the absolute coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

param marker
optional repeated capability selector. Default value: Nr1

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

6.1.1.3.5.65 Relative

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:MARKer<mnr>:RELative
```

class RelativeCls

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

class FetchStruct

Response structure. Fields:

- Dx_Value: float: Delta X value of the marker relative to the reference marker Unit: Hz
- Dy_Value: float: Delta Y value of the marker relative to the reference marker Unit: Depends on input path and demodulation type

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *markerOther=MarkerOther.Nr2*) → FetchStruct

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:MARKer<mnr>
↪:RELative
value: FetchStruct = driver.afRf.measurement.multiEval.fft.voip.marker.relative.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪markerOther = repcap.MarkerOther.Nr2)
```

Query the relative coordinates of marker number <mnr>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

param markerOther
optional repeated capability selector. Default value: Nr2

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

6.1.1.3.5.66 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.afRf.measurement.multiEval.fft.voip.power.clone()
```

Subgroups

6.1.1.3.5.67 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.average.
↪fetch()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:AVERage
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.average.
↪read()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.68 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.current.
↪fetch()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:CURRent
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.current.
↪read()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.69 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.maximum.
↪fetch()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MAXimum
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.maximum.
↪read()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.5.70 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MINimum
```

class MinimumCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.minimum.
↪fetch()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:FFT:VOIP:POWer:MINimum
value: List[float] = driver.afRf.measurement.multiEval.fft.voip.power.minimum.
↪read()
```

Query the contents of the spectrum diagram for the VoIP path.

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

return
power: Comma-separated list of 1793 audio level values (diagram from left to right)
Unit: %

6.1.1.3.6 Oscilloscope

class OscilloscopeCls

Oscilloscope commands group definition. 20 total commands, 7 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.6.1 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.afRf.measurement.multiEval.oscilloscope.audioInput.repcap_audioInput_get()
driver.afRf.measurement.multiEval.oscilloscope.audioInput.repcap_audioInput_set(repcap.
↪AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.oscilloscope.audioInput.clone()
```

Subgroups

6.1.1.3.6.2 PowerVsTime

class PowerVsTimeCls

PowerVsTime 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.afRf.measurement.multiEval.oscilloscope.audioInput.powerVsTime.clone()
```

Subgroups

6.1.1.3.6.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN<Nr>:PVTime:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN<Nr>:PVTime:CURRent
```

class CurrentCls

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

fetch(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN<Nr>
↪:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.audioInput.
↪powerVsTime.current.fetch(audioInput = repcap.AudioInput.Default)
```

Query the contents of the AF oscilloscope diagram for an AF input path.

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

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: V

read(*audioInput=AudioInput.Default*) → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN<Nr>
↪:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.audioInput.
↪powerVsTime.current.read(audioInput = repcap.AudioInput.Default)
```

Query the contents of the AF oscilloscope diagram for an AF input path.

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

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: V

6.1.1.3.6.4 DemodLeft

class DemodLeftCls

DemodLeft 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.afRf.measurement.multiEval.oscilloscope.demodLeft.clone()
```

Subgroups

6.1.1.3.6.5 Fdeviation

class FdeviationCls

Fdeviation 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.afRf.measurement.multiEval.oscilloscope.demodLeft.fdeviation.clone()
```

Subgroups

6.1.1.3.6.6 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEMLeft:FDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEMLeft:FDEViation:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEMLeft:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.demodLeft.
↪fdeviation.current.fetch()
```

Query the contents of the AF oscilloscope diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, the results are related to the left audio channel.

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

return
dvt_time: Comma-separated list of 960 frequency deviation or audio deviation values (diagram from left to right) Unit: Hz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEMLeft:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.demodLeft.
↪fdeviation.current.read()
```

Query the contents of the AF oscilloscope diagram for the left demodulator channel and FM demodulation or FM stereo demodulation. For FM stereo, the results are related to the left audio channel.

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

return
dvt_time: Comma-separated list of 960 frequency deviation or audio deviation values (diagram from left to right) Unit: Hz

6.1.1.3.6.7 DemodRight

class DemodRightCls

DemodRight 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.afRf.measurement.multiEval.oscilloscope.demodRight.clone()
```

Subgroups

6.1.1.3.6.8 Fdeviation

class FdeviationCls

Fdeviation 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.afRf.measurement.multiEval.oscilloscope.demodRight.fdeviation.clone()
```

Subgroups

6.1.1.3.6.9 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEMRight:FDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEMRight:FDEViation:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEMRight:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.demodRight.
↪fdeviation.current.fetch()
```

Query the contents of the AF oscilloscope diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
dvt_time: Comma-separated list of 960 audio deviation values (diagram from left to right) Unit: Hz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEMRight:FDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.demodRight.
↪fdeviation.current.read()
```

Query the contents of the AF oscilloscope diagram for the right demodulator channel and FM stereo demodulation. The results are related to the right audio channel.

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

return
dvt_time: Comma-separated list of 960 audio deviation values (diagram from left to right) Unit: Hz

6.1.1.3.6.10 Demodulation

class DemodulationCls

Demodulation 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.afRf.measurement.multiEval.oscilloscope.demodulation.clone()
```

Subgroups

6.1.1.3.6.11 LsbLevel

class LsbLevelCls

LsbLevel 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.afRf.measurement.multiEval.oscilloscope.demodulation.lsbLevel.clone()
```

Subgroups

6.1.1.3.6.12 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:LSBLevel:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:LSBLevel:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:LSBLevel:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.lsbLevel.current.fetch()
```

Query the contents of the AF oscilloscope diagram for SSB-LSB demodulation.

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

return
dvt_time: Comma-separated list of 960 audio level values (diagram from left to right)
Unit: V

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:LSBLevel:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.lsbLevel.current.read()
```

Query the contents of the AF oscilloscope diagram for SSB-LSB demodulation.

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

return
dvt_time: Comma-separated list of 960 audio level values (diagram from left to right)
Unit: V

6.1.1.3.6.13 ModDepth

class ModDepthCls

ModDepth 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.afRf.measurement.multiEval.oscilloscope.demodulation.modDepth.clone()
```

Subgroups

6.1.1.3.6.14 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:MDEPth:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:MDEPth:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MDEPth:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.modDepth.current.fetch()
```

Query the contents of the AF oscilloscope diagram for AM demodulation.

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

return
dvt_time: Comma-separated list of 960 modulation depth values (diagram from left to right) Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MDEPth:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.modDepth.current.read()
```

Query the contents of the AF oscilloscope diagram for AM demodulation.

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

return
dvt_time: Comma-separated list of 960 modulation depth values (diagram from left to right) Unit: %

6.1.1.3.6.15 Pdeviation

class PdeviationCls

Pdeviation 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.afRf.measurement.multiEval.oscilloscope.demodulation.pdeviation.clone()
```

Subgroups

6.1.1.3.6.16 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:PDEViation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:PDEViation:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:PDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.pdeviation.current.fetch()
```

Query the contents of the AF oscilloscope diagram for PM demodulation.

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

return
dvt_time: Comma-separated list of 960 phase-deviation values (diagram from left to right) Unit: rad

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:PDEViation:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.pdeviation.current.read()
```

Query the contents of the AF oscilloscope diagram for PM demodulation.

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

return
dvt_time: Comma-separated list of 960 phase-deviation values (diagram from left to right) Unit: rad

6.1.1.3.6.17 UsbLevel

class UsbLevelCls

UsbLevel 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.afRf.measurement.multiEval.oscilloscope.demodulation.usbLevel.clone()
```

Subgroups

6.1.1.3.6.18 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:USBLevel:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:USBLevel:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:USBLevel:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.usbLevel.current.fetch()
```

Query the contents of the AF oscilloscope diagram for SSB-USB demodulation.

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

return
dvt_time: Comma-separated list of 960 audio level values (diagram from left to right)
Unit: V

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:USBLevel:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.
↪demodulation.usbLevel.current.read()
```

Query the contents of the AF oscilloscope diagram for SSB-USB demodulation.

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

return
dvt_time: Comma-separated list of 960 audio level values (diagram from left to right)
Unit: V

6.1.1.3.6.19 SpdifLeft

class SpdifLeftCls

SpdifLeft 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.afRf.measurement.multiEval.oscilloscope.spdifLeft.clone()
```

Subgroups

6.1.1.3.6.20 PowerVsTime

class PowerVsTimeCls

PowerVsTime 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.afRf.measurement.multiEval.oscilloscope.spdifLeft.powerVsTime.clone()
```

Subgroups

6.1.1.3.6.21 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SINLeft:PVTime:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SINLeft:PVTime:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SINLeft:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.spdifLeft.
↪powerVsTime.current.fetch()
```

Query the contents of the AF oscilloscope diagram for the left or right SPDIF channel.

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: % full scale

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SINLeft:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.spdifLeft.
↪powerVsTime.current.read()
```

Query the contents of the AF oscilloscope diagram for the left or right SPDIF channel.

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: % full scale

6.1.1.3.6.22 SpdifRight

class SpdifRightCls

SpdifRight 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.afRf.measurement.multiEval.oscilloscope.spdifRight.clone()
```

Subgroups

6.1.1.3.6.23 PowerVsTime

class PowerVsTimeCls

PowerVsTime 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.afRf.measurement.multiEval.oscilloscope.spdifRight.powerVsTime.clone()
```

Subgroups

6.1.1.3.6.24 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SINRight:PVTime:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SINRight:PVTime:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SINRight:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.spdifRight.
↪powerVsTime.current.fetch()
```

Query the contents of the AF oscilloscope diagram for the left or right SPDIF channel.

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: % full scale

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SINRight:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.spdifRight.
↪powerVsTime.current.read()
```

Query the contents of the AF oscilloscope diagram for the left or right SPDIF channel.

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: % full scale

6.1.1.3.6.25 Voip

class VoipCls

Voip 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.afRf.measurement.multiEval.oscilloscope.voip.clone()
```

Subgroups

6.1.1.3.6.26 PowerVsTime

class PowerVsTimeCls

PowerVsTime 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.afRf.measurement.multiEval.oscilloscope.voip.powerVsTime.clone()
```

Subgroups

6.1.1.3.6.27 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:PVTime:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:PVTime:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.voip.
↪powerVsTime.current.fetch()
```

Query the contents of the AF oscilloscope diagram for the VoIP path.

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: % full scale

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:PVTime:CURRent
value: List[float] = driver.afRf.measurement.multiEval.oscilloscope.voip.
↪powerVsTime.current.read()
```

Query the contents of the AF oscilloscope diagram for the VoIP path.

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

return
power_vs_time: Comma-separated list of 960 audio level values (diagram from left to right) Unit: % full scale

6.1.1.3.7 RfCarrier

class RfCarrierCls

RfCarrier commands group definition. 61 total commands, 9 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.7.1 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: enums.ResultStatus: Carrier frequency error Unit: Hz
- Power_Rms: enums.ResultStatus: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float or bool: Absolute peak envelope power of the RF signal Unit: dBm

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: float: Carrier frequency error Unit: Hz
- Power_Rms: float: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float: Absolute peak envelope power of the RF signal Unit: dBm

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.rfCarrier.average.
↪calculate()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

return

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:AVERage
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.average.fetch()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:AVERage
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.average.read()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.7.2 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: enums.ResultStatus: Carrier frequency error Unit: Hz
- Power_Rms: enums.ResultStatus: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float or bool: Absolute peak envelope power of the RF signal Unit: dBm

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: float: Carrier frequency error Unit: Hz
- Power_Rms: float: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float: Absolute peak envelope power of the RF signal Unit: dBm

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.rfCarrier.current.
↪calculate()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:CURRent
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.current.fetch()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:CURRent
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.current.read()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.7.3 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:DEViation
```

class DeviationCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: enums.ResultStatus: Carrier frequency error Unit: Hz
- Power_Rms: enums.ResultStatus: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float or bool: Absolute peak envelope power of the RF signal Unit: dBm

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: float: Carrier frequency error Unit: Hz
- Power_Rms: float: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float: Absolute peak envelope power of the RF signal Unit: dBm

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.rfCarrier.deviation.
↪calculate()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:DEViation
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.deviation.
↪fetch()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:DEViation
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.deviation.read()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.7.4 FreqError

class FreqErrorCls

FreqError commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.rfCarrier.freqError.clone()
```

Subgroups

6.1.1.3.7.5 Delta

class DeltaCls

Delta commands group definition. 10 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.clone()
```

Subgroups

6.1.1.3.7.6 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪average.fetch()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪average.read()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

6.1.1.3.7.7 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪current.fetch()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪current.read()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

6.1.1.3.7.8 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪deviation.fetch()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪deviation.read()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

6.1.1.3.7.9 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪maximum.fetch()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪maximum.read()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

6.1.1.3.7.10 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FERRor:DELTa:MINimum
```

class MinimumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪minimum.fetch()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FERRor:DELTa:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.freqError.delta.
↪minimum.read()
```

Queries delta results for carrier frequency error.

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

return
freq_error: Carrier frequency error Unit: Hz

6.1.1.3.7.11 Frequency

class FrequencyCls

Frequency commands group definition. 16 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.7.12 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.average.
↪fetch()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.average.
↪read()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.13 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.current.
↪fetch()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.current.
↪read()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.14 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.rfCarrier.frequency.delta.clone()
```

Subgroups

6.1.1.3.7.15 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪average.fetch()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪average.read()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.16 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪current.fetch()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪current.read()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.17 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪deviation.fetch()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪deviation.read()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.18 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪maximum.fetch()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.delta.
↪maximum.read()
```

Queries delta results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.19 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.maximum.
↪fetch()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.maximum.
↪read()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.20 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MINimum
```

class MinimumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.minimum.
↪fetch()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:FREQuency:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.frequency.minimum.
↪read()
```

Queries results for carrier frequency.

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

return
frequency: Carrier frequency Unit: Hz

6.1.1.3.7.21 Maximum

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MAXimum
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MAXimum
```

class MaximumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: enums.ResultStatus: Carrier frequency error Unit: Hz
- Power_Rms: enums.ResultStatus: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float or bool: Absolute peak envelope power of the RF signal Unit: dBm

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: float: Carrier frequency error Unit: Hz
- Power_Rms: float: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float: Absolute peak envelope power of the RF signal Unit: dBm

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MAXimum
value: CalculateStruct = driver.afRf.measurement.multiEval.rfCarrier.maximum.
↪calculate()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.maximum.fetch()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.maximum.read()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.7.22 Minimum

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MINimum
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MINimum
```

class MinimumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: enums.ResultStatus: Carrier frequency error Unit: Hz
- Power_Rms: enums.ResultStatus: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float or bool: Absolute peak envelope power of the RF signal Unit: dBm

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: float: Carrier frequency error Unit: Hz
- Power_Rms: float: Absolute RMS power of the RF signal Unit: dBm
- Power_Pep: float: Absolute peak envelope power of the RF signal Unit: dBm

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MINimum
value: CalculateStruct = driver.afRf.measurement.multiEval.rfCarrier.minimum.
↪calculate()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MINimum
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.minimum.fetch()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:MINimum
value: ResultData = driver.afRf.measurement.multiEval.rfCarrier.minimum.read()
```

Queries the RF carrier measurement results. CALCulate commands return error indicators instead of measurement values.

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

6.1.1.3.7.23 PePower

class PePowerCls

PePower commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.rfCarrier.pePower.clone()
```

Subgroups

6.1.1.3.7.24 Delta

class DeltaCls

Delta commands group definition. 10 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.clone()
```

Subgroups

6.1.1.3.7.25 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪average.fetch()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪average.read()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

6.1.1.3.7.26 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪current.fetch()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪current.read()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

6.1.1.3.7.27 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪deviation.fetch()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪deviation.read()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

6.1.1.3.7.28 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪maximum.fetch()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪maximum.read()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

6.1.1.3.7.29 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:PEPower:DELTa:MINimum
```

class MinimumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪minimum.fetch()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:PEPower:DELTa:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.pePower.delta.
↪minimum.read()
```

Queries delta results for carrier power PEP value.

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

return
power_pep: Unit: dB

6.1.1.3.7.30 Power

class PowerCls

Power commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.7.31 Delta

class DeltaCls

Delta commands group definition. 10 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.rfCarrier.power.delta.clone()
```

Subgroups

6.1.1.3.7.32 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:AVERage
```

class AverageCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.average.
↪fetch()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.average.
↪read()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

6.1.1.3.7.33 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:CURRent
```

class CurrentCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.current.
↪fetch()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.current.
↪read()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

6.1.1.3.7.34 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:DEViation
```

class DeviationCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.
↪deviation.fetch()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.
↪deviation.read()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

6.1.1.3.7.35 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:MAXimum
```

class MaximumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.maximum.
↪fetch()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.maximum.
↪read()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

6.1.1.3.7.36 Minimum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:MINimum
READ:AFRF:MEASurement<Instance>:MEValuation:RFCarrier:POWer:DELTa:MINimum
```

class MinimumCls

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

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.minimum.
↪fetch()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:RFCarrier:POWer:DELTa:MINimum
value: float = driver.afRf.measurement.multiEval.rfCarrier.power.delta.minimum.
↪read()
```

Queries delta results for carrier power RMS value.

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

return
power_rms: Unit: dB

6.1.1.3.8 SignalQuality

class SignalQualityCls

SignalQuality commands group definition. 72 total commands, 6 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.3.8.1 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.afRf.measurement.multiEval.signalQuality.audioInput.repcap_audioInput_get()
driver.afRf.measurement.multiEval.signalQuality.audioInput.repcap_audioInput_set(repcap.
↪AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 12 total commands, 4 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.signalQuality.audioInput.clone()
```

Subgroups

6.1.1.3.8.2 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate(*audioInput=AudioInput.Default*) → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>
↪:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪audioInput.average.calculate(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

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

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

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪average.fetch(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

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

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

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪average.read(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

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

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

6.1.1.3.8.3 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate(*audioInput=AudioInput.Default*) → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>
↪:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪audioInput.current.calculate(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

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

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

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪current.fetch(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

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

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

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪current.read(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

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

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

6.1.1.3.8.4 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:DEViation
```

class DeviationCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'

- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate(*audioInput=AudioInput.Default*) → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>
↪:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪audioInput.deviation.calculate(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪deviation.fetch(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪deviation.read(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.5 Extreme

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:EXTReme
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:EXTReme
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:EXTReme
```

class ExtremeCls

Extreme commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB

- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate(*audioInput=AudioInput.Default*) → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>
↪:EXTReme
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪audioInput.extreme.calculate(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪extreme.fetch(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:AIN<Nr>:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.audioInput.
↪extreme.read(audioInput = repcap.AudioInput.Default)
```

Query the signal quality results measured for an AF input path. CALCulate commands return error indicators instead of measurement values.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.6 DemodLeft

class DemodLeftCls

DemodLeft commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.signalQuality.demodLeft.clone()
```

Subgroups

6.1.1.3.8.7 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB

- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMLeft:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodLeft.average.calculate()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪average.fetch()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪average.read()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.8 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMLeft:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodLeft.current.calculate()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪current.fetch()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪current.read()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.9 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:DEViation
```

class DeviationCls

Deviation commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMLeft:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodLeft.deviation.calculate()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪deviation.fetch()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪deviation.read()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.10 Extreme

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:EXTReme
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:EXTReme
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:EXTReme
```

class ExtremeCls

Extreme commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'

- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMLeft:EXTReme
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodLeft.extreme.calculate()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪extreme.fetch()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeft:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodLeft.
↪extreme.read()
```

Query the signal quality results measured for the left demodulator channel. For FM stereo, these results are related to the left audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.11 DemodRight

class DemodRightCls

DemodRight commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.signalQuality.demodRight.clone()
```

Subgroups

6.1.1.3.8.12 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMRight:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodRight.average.calculate()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪average.fetch()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪average.read()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.13 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMRight:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodRight.current.calculate()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪current.fetch()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪current.read()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.14 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:DEViation
```

class DeviationCls

Deviation commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMRight:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodRight.deviation.calculate()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪deviation.fetch()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪deviation.read()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.15 Extreme

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:EXTReme
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:EXTReme
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:EXTReme
```

class ExtremeCls

Extreme commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:DEMRight:EXTReme
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪demodRight.extreme.calculate()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪extreme.fetch()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRight:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.demodRight.
↪extreme.read()
```

Query the signal quality results measured for the right demodulator channel. The commands are only relevant for FM stereo. The results are related to the right audio channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.16 SpdifLeft

class SpdifLeftCls

SpdifLeft commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.clone()
```

Subgroups

6.1.1.3.8.17 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINLeft:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifLeft.average.calculate()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪average.fetch()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪average.read()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.18 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINLeft:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifLeft.current.calculate()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪current.fetch()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪current.read()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.19 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:DEViation
```

class DeviationCls

Deviation commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB

- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINLeft:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifLeft.deviation.calculate()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪deviation.fetch()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪deviation.read()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.20 Extreme

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:EXTReme
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:EXTReme
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:EXTReme
```

class ExtremeCls

Extreme commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINLeft:EXTReme
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifLeft.extreme.calculate()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪extreme.fetch()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeft:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifLeft.
↪extreme.read()
```

Query the signal quality results measured for the left SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.21 SpdifRight

class SpdifRightCls

SpdifRight commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.signalQuality.spdifRight.clone()
```

Subgroups

6.1.1.3.8.22 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINRight:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifRight.average.calculate()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪average.fetch()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪average.read()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.23 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINRight:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifRight.current.calculate()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪current.fetch()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪current.read()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.24 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:DEViation
```

class DeviationCls

Deviation commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB

- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINRight:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifRight.deviation.calculate()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪deviation.fetch()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪deviation.read()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.25 Extreme

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:EXTReme
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:EXTReme
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:EXTReme
```

class ExtremeCls

Extreme commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>
↪:MEValuation:SQUality:SINRight:EXTReme
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.
↪spdifRight.extreme.calculate()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪extreme.fetch()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRight:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.spdifRight.
↪extreme.read()
```

Query the signal quality results measured for the right SPDIF channel. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.26 Voip

class VoipCls

Voip commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.signalQuality.voip.clone()
```

Subgroups

6.1.1.3.8.27 Average

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:AVERage
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:AVERage
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.voip.
↪average.calculate()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪average.fetch()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:AVERage
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪average.read()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.28 Current

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:CURRent
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:CURRent
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.voip.
↪current.calculate()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪current.fetch()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:CURRent
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪current.read()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.29 Deviation

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:DEViation
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:DEViation
```

class DeviationCls

Deviation commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %

- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:DEViation
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.voip.
↪deviation.calculate()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪deviation.fetch()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:DEViation
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪deviation.read()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.8.30 Extreme

SCPI Command:

```
CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:EXTReme
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:EXTReme
READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:EXTReme
```

class ExtremeCls

Extreme commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: enums.ResultStatus: Total harmonic distortion in percent Unit: %
- Thd: float or bool: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float or bool: Total harmonic distortion and noise Unit: %
- Sinad: enums.ResultStatus: Signal to noise and distortion Unit: dB
- Snr: enums.ResultStatus: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float or bool: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float or bool: (S+N+D) /N Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Thd_Percent: float: Total harmonic distortion in percent Unit: %
- Thd: float: Total harmonic distortion in dB Unit: dB
- Thd_Plus_Noise: float: Total harmonic distortion and noise Unit: %
- Sinad: float: Signal to noise and distortion Unit: dB
- Snr: float: Signal-to-noise ratio S/N Unit: dB
- Snr_Plus_Noise: float: (S+N) /N Unit: dB
- Snr_Plus_Noinse_Plus_Dist: float: (S+N+D) /N Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:EXTReme
value: CalculateStruct = driver.afRf.measurement.multiEval.signalQuality.voip.
↪extreme.calculate()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪extreme.fetch()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIP:EXTReme
value: ResultData = driver.afRf.measurement.multiEval.signalQuality.voip.
↪extreme.read()
```

Query the signal quality results measured for the VoIP path. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9 SpdifLeft

class SpdifLeftCls

SpdifLeft commands group definition. 24 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.spdifLeft.clone()
```

Subgroups

6.1.1.3.9.1 AfSignal

class AfSignalCls

AfSignal 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.afRf.measurement.multiEval.spdifLeft.afSignal.clone()
```

Subgroups

6.1.1.3.9.2 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪average.fetch()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪average.read()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪current.fetch()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪current.read()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9.4 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪deviation.fetch()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪deviation.read()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9.5 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪maximum.fetch()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.afSignal.
↪maximum.read()
```

Query the AF frequency and level results for the left SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9.6 Frequency

class FrequencyCls

Frequency commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.spdifLeft.frequency.clone()
```

Subgroups

6.1.1.3.9.7 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.spdifLeft.frequency.delta.clone()
```

Subgroups

6.1.1.3.9.8 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪average.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪average.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.9.9 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪current.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪current.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.9.10 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪deviation.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪deviation.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.9.11 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:FREQuency:DELTa:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪maximum.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.spdifLeft.frequency.delta.
↪maximum.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.9.12 Level

class LevelCls

Level commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.spdifLeft.level.clone()
```

Subgroups

6.1.1.3.9.13 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.spdifLeft.level.delta.clone()
```

Subgroups

6.1.1.3.9.14 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪average.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪average.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9.15 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪current.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪current.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9.16 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:LEVel:DELTa:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪deviation.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINLeft:LEVel:DELTa:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪deviation.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.9.17 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪maximum.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINLeft:LEVel:DELTa:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifLeft.level.delta.
↪maximum.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10 SpdifRight

class SpdifRightCls

SpdifRight commands group definition. 24 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.spdifRight.clone()
```

Subgroups

6.1.1.3.10.1 AfSignal

class AfSignalCls

AfSignal 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.afRf.measurement.multiEval.spdifRight.afSignal.clone()
```

Subgroups

6.1.1.3.10.2 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Range: 0 Hz to 21 kHz, Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪average.fetch()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪average.read()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Range: 0 Hz to 21 kHz, Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪current.fetch()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪current.read()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10.4 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Range: 0 Hz to 21 kHz, Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪deviation.fetch()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪deviation.read()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10.5 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Range: 0 Hz to 21 kHz, Unit: Hz
- Level_Peak: float: Effective level of the AC component of the measured AF signal Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪maximum.fetch()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.afSignal.
↪maximum.read()
```

Query the AF frequency and level results for the right SPDIF channel.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10.6 Frequency

class FrequencyCls

Frequency commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.spdifRight.frequency.clone()
```

Subgroups

6.1.1.3.10.7 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.spdifRight.frequency.delta.clone()
```

Subgroups

6.1.1.3.10.8 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪average.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪average.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.10.9 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪current.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪current.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.10.10 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪deviation.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪deviation.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.10.11 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:FREQuency:DELTa:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪maximum.fetch()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.spdifRight.frequency.delta.
↪maximum.read()
```

Query delta results for AF frequency of SPDIF path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.10.12 Level

class LevelCls

Level commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.spdifRight.level.clone()
```

Subgroups

6.1.1.3.10.13 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.spdifRight.level.delta.clone()
```

Subgroups

6.1.1.3.10.14 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:LEVel:DELTa:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪average.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:AVERage
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪average.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10.15 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:LEVel:DELTa:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪current.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:CURRent
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪current.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10.16 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:LEVel:DELTa:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪deviation.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:LEVel:DELTa:DEViation
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪deviation.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.10.17 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Unit: %
- Level_Rms: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:SINRight:LEVel:DELTa:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪maximum.fetch()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:SINRight:LEVel:DELTa:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.spdifRight.level.delta.
↪maximum.read()
```

Query delta results for AF frequency of SPDIF path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.11 State

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:STATe
value: enums.ResourceState = driver.afRf.measurement.multiEval.state.fetch()
```

Queries the main analyzer state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.state.clone()
```

Subgroups

6.1.1.3.11.1 All

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[ResourceState]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:STATe:ALL
value: List[enums.ResourceState] = driver.afRf.measurement.multiEval.state.all.
↪fetch()
```

Queries the main analyzer state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.1.1.3.12 Tones

class TonesCls

Tones commands group definition. 54 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.tones.clone()
```

Subgroups

6.1.1.3.12.1 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.afRf.measurement.multiEval.tones.audioInput.repcap_audioInput_get()
driver.afRf.measurement.multiEval.tones.audioInput.repcap_audioInput_set(repcap.
↪AudioInput.Nr1)
```

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>
```

class AudioInputCls

AudioInput commands group definition. 6 total commands, 2 Subgroups, 2 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Dialed digit as string
- Frequency_1: List[float]: Nominal tone frequency according to the tone table Unit: Hz
- Deviation_1: List[float]: Deviation of the measured tone frequency from the nominal tone frequency Unit: Hz
- Fequency_2: List[float]: Second nominal frequency (only relevant for dual tones) Unit: Hz
- Deviation_2: List[float]: Deviation of the second frequency (only relevant for dual tones) Unit: Hz
- Time: List[float]: Measured tone duration Unit: s
- Pause: List[float]: Duration of the pause between this tone and the next tone of the sequence Unit: s

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>
value: ResultData = driver.afRf.measurement.multiEval.tones.audioInput.
↪fetch(audioInput = repcap.AudioInput.Default)
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>
value: ResultData = driver.afRf.measurement.multiEval.tones.audioInput.
↪read(audioInput = repcap.AudioInput.Default)
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.tones.audioInput.clone()
```

Subgroups

6.1.1.3.12.2 Repetitions

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:REPetitions
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:REPetitions
```

class RepetitionsCls

Repetitions commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch(*audioInput=AudioInput.Default*) → int

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.audioInput.repetitions.
↪fetch(audioInput = repcap.AudioInput.Default)
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
repetitions: No help available

read(*audioInput=AudioInput.Default*) → int

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.audioInput.repetitions.
↪read(audioInput = repcap.AudioInput.Default)
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
repetitions: No help available

6.1.1.3.12.3 Sequence

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:SEQuence
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:SEQuence
```

class SequenceCls

Sequence commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Comma-separated list of 42 strings Each string indicates a dialed digit.

fetch(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.audioInput.sequence.
↪fetch(audioInput = repcap.AudioInput.Default)
```

Query the digit results of a tone sequence analysis.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

read(*audioInput=AudioInput.Default*) → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.audioInput.sequence.
↪read(audioInput = repcap.AudioInput.Default)
```

Query the digit results of a tone sequence analysis.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.12.4 Dcs

class DcsCls

Dcs commands group definition. 24 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.tones.dcs.clone()
```

Subgroups

6.1.1.3.12.5 BitErrorRate

class BitErrorRateCls

BitErrorRate 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.afRf.measurement.multiEval.tones.dcs.bitErrorRate.clone()
```

Subgroups

6.1.1.3.12.6 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:AVERage
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.average.
↪fetch()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:AVERage
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.average.
↪read()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

6.1.1.3.12.7 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:CURRent
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.current.
↪fetch()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:CURRent
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.current.
↪read()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

6.1.1.3.12.8 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:DEViation
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.
↪deviation.fetch()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:DEViation
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.
↪deviation.read()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

6.1.1.3.12.9 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:MAXimum
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.maximum.
↪fetch()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:BERate:MAXimum
value: float = driver.afRf.measurement.multiEval.tones.dcs.bitErrorRate.maximum.
↪read()
```

Query the bit error rate measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bit_error_rate: Number of bit errors per second

6.1.1.3.12.10 Cword

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:CWORd
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:CWORd
```

class CwordCls

Cword commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[str]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:CWORd
value: List[str] = driver.afRf.measurement.multiEval.tones.dcs.cword.fetch()
```

Query the code number of the five last detected code words. Code word results are separated by commas.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
codeword: Detected DCS code number as octal number Range: 16 to 511

read() → List[str]

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:CWORd
value: List[str] = driver.afRf.measurement.multiEval.tones.dcs.cword.read()
```

Query the code number of the five last detected code words. Code word results are separated by commas.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
codeword: Detected DCS code number as octal number Range: 16 to 511

6.1.1.3.12.11 Dmatches

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:DMATches
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:DMATches
```

class DmatchesCls

Dmatches commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → int

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:DMATches
value: int = driver.afRf.measurement.multiEval.tones.dcs.dmatches.fetch()
```

Query the number of received code words that matched the expected code word.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
detected_matches: Number of matches

read() → int

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:DMATches
value: int = driver.afRf.measurement.multiEval.tones.dcs.dmatches.read()
```

Query the number of received code words that matched the expected code word.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
detected_matches: Number of matches

6.1.1.3.12.12 FskDeviation

class FskDeviationCls

FskDeviation 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.afRf.measurement.multiEval.tones.dcs.fskDeviation.clone()
```

Subgroups

6.1.1.3.12.13 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:AVERage
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.average.
↪fetch()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:AVERage
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.average.
↪read()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

6.1.1.3.12.14 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:CURRent
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.current.
↪fetch()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:CURRent
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.current.
↪read()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

6.1.1.3.12.15 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:DEViation
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.
↪deviation.fetch()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:DEViation
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.
↪deviation.read()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

6.1.1.3.12.16 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:FSKDeviation:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:MAXimum
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.maximum.
↪fetch()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DCS:FSKDeviation:MAXimum
value: float = driver.afRf.measurement.multiEval.tones.dcs.fskDeviation.maximum.
↪read()
```

Query the FSK deviation measured for a DCS signal.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

6.1.1.3.12.17 LcWord

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:LCWord
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:LCWord
```

class LcWordCls

LcWord commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → str

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:LCWord
value: str = driver.afRf.measurement.multiEval.tones.dcs.lcWord.fetch()
```

Query the code number of the last detected code word that matched the expected code word.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
last_code_word: Detected DCS code number as octal number

read() → str

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:LCWord
value: str = driver.afRf.measurement.multiEval.tones.dcs.lcWord.read()
```

Query the code number of the last detected code word that matched the expected code word.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
last_code_word: Detected DCS code number as octal number

6.1.1.3.12.18 TocLength

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:TOCLength
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:TOCLength
```

class TocLengthCls

TocLength commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:TOCLength
value: float = driver.afRf.measurement.multiEval.tones.dcs.tocLength.fetch()
```

Query the duration of the last received turn-off code.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
off_code_length: Unit: s

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:TOCLength
value: float = driver.afRf.measurement.multiEval.tones.dcs.tocLength.read()
```

Query the duration of the last received turn-off code.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
off_code_length: Unit: s

6.1.1.3.12.19 Demodulation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation
```

class DemodulationCls

Demodulation commands group definition. 6 total commands, 2 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Dialed digit as string
- Frequency_1: List[float]: Nominal tone frequency according to the tone table Unit: Hz
- Deviation_1: List[float]: Deviation of the measured tone frequency from the nominal tone frequency Unit: Hz
- Fequency_2: List[float]: Second nominal frequency (only relevant for dual tones) Unit: Hz
- Deviation_2: List[float]: Deviation of the second frequency (only relevant for dual tones) Unit: Hz
- Time: List[float]: Measured tone duration Unit: s
- Pause: List[float]: Duration of the pause between this tone and the next tone of the sequence Unit: s

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation
value: ResultData = driver.afRf.measurement.multiEval.tones.demodulation.fetch()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation
value: ResultData = driver.afRf.measurement.multiEval.tones.demodulation.read()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.tones.demodulation.clone()
```

Subgroups

6.1.1.3.12.20 Repetitions

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:REPetitions
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:REPetitions
```

class RepetitionsCls

Repetitions commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → int

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DEModulation:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.demodulation.repetitions.
↪fetch()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

read() → int

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DEModulation:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.demodulation.repetitions.
↪read()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

6.1.1.3.12.21 Sequence

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:SEQuence
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:SEQuence
```

class SequenceCls

Sequence commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Comma-separated list of 42 strings Each string indicates a dialed digit.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.demodulation.
↪sequence.fetch()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.demodulation.
↪sequence.read()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.12.22 SpdifLeft

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft
```

class SpdifLeftCls

SpdifLeft commands group definition. 6 total commands, 2 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'

- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Dialed digit as string
- Frequency_1: List[float]: Nominal tone frequency according to the tone table Unit: Hz
- Deviation_1: List[float]: Deviation of the measured tone frequency from the nominal tone frequency Unit: Hz
- Fequency_2: List[float]: Second nominal frequency (only relevant for dual tones) Unit: Hz
- Deviation_2: List[float]: Deviation of the second frequency (only relevant for dual tones) Unit: Hz
- Time: List[float]: Measured tone duration Unit: s
- Pause: List[float]: Duration of the pause between this tone and the next tone of the sequence Unit: s

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifLeft.fetch()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifLeft.read()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.tones.spdifLeft.clone()
```

Subgroups

6.1.1.3.12.23 Repetitions

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:REPetitions
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:REPetitions
```

class RepetitionsCls

Repetitions commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → int

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.spdifLeft.repetitions.
↪fetch()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

read() → int

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.spdifLeft.repetitions.
↪read()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

6.1.1.3.12.24 Sequence

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:SEQuence
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:SEQuence
```

class SequenceCls

Sequence commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Comma-separated list of 42 strings Each string indicates a dialed digit.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifLeft.sequence.
↪fetch()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifLeft.sequence.
↪read()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.12.25 SpdifRight

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight
```

class SpdifRightCls

SpdifRight commands group definition. 6 total commands, 2 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Dialed digit as string
- Frequency_1: List[float]: Nominal tone frequency according to the tone table Unit: Hz
- Deviation_1: List[float]: Deviation of the measured tone frequency from the nominal tone frequency Unit: Hz
- Fequency_2: List[float]: Second nominal frequency (only relevant for dual tones) Unit: Hz
- Deviation_2: List[float]: Deviation of the second frequency (only relevant for dual tones) Unit: Hz
- Time: List[float]: Measured tone duration Unit: s
- Pause: List[float]: Duration of the pause between this tone and the next tone of the sequence Unit: s

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifRight.fetch()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifRight.read()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.tones.spdifRight.clone()
```

Subgroups

6.1.1.3.12.26 Repetitions

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:REPetitions
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:REPetitions
```

class RepetitionsCls

Repetitions commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → int

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.spdifRight.repetitions.
↪fetch()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

read() → int

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.spdifRight.repetitions.
↪read()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

6.1.1.3.12.27 Sequence

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:SEQuence
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:SEQuence
```

class SequenceCls

Sequence commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Comma-separated list of 42 strings Each string indicates a dialed digit.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifRight.sequence.
↪fetch()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.spdifRight.sequence.
↪read()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.12.28 Voip

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP
```

class VoipCls

Voip commands group definition. 6 total commands, 2 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'

- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Dialed digit as string
- Frequency_1: List[float]: Nominal tone frequency according to the tone table Unit: Hz
- Deviation_1: List[float]: Deviation of the measured tone frequency from the nominal tone frequency Unit: Hz
- Fequency_2: List[float]: Second nominal frequency (only relevant for dual tones) Unit: Hz
- Deviation_2: List[float]: Deviation of the second frequency (only relevant for dual tones) Unit: Hz
- Time: List[float]: Measured tone duration Unit: s
- Pause: List[float]: Duration of the pause between this tone and the next tone of the sequence Unit: s

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP
value: ResultData = driver.afRf.measurement.multiEval.tones.voip.fetch()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP
value: ResultData = driver.afRf.measurement.multiEval.tones.voip.read()
```

Query all results of a tone sequence analysis. For each tone, a sequence of results is returned: <Reliability>, <Length>{, <Sequence>, <Frequency1>, <Deviation1>, <Frequency2>, <Deviation2>, <Time>, <Pause>}Tone 1, {...}Tone 2, ..., {...}Tone <Length>

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.tones.voip.clone()
```

Subgroups

6.1.1.3.12.29 Repetitions

SCPI Command:

```
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:REPetitions
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:REPetitions
```

class RepetitionsCls

Repetitions commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → int

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.voip.repetitions.fetch()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

read() → int

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:REPetitions
value: int = driver.afRf.measurement.multiEval.tones.voip.repetitions.read()
```

Query the number of measured tone sequences.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
repetitions: No help available

6.1.1.3.12.30 Sequence

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:SEQuence
READ:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:SEQuence
```

class SequenceCls

Sequence commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Length: int: Length of the tone sequence (number of digits)
- Sequence: List[str]: Comma-separated list of 42 strings Each string indicates a dialed digit.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.voip.sequence.
↪fetch()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:SEQuence
value: ResultData = driver.afRf.measurement.multiEval.tones.voip.sequence.read()
```

Query the digit results of a tone sequence analysis.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13 Voip

class VoipCls

Voip commands group definition. 24 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.voip.clone()
```

Subgroups

6.1.1.3.13.1 AfSignal

class AfSignalCls

AfSignal 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.afRf.measurement.multiEval.voip.afSignal.clone()
```

Subgroups

6.1.1.3.13.2 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.average.
↪fetch()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:AVERage
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.average.
↪read()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13.3 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.current.
↪fetch()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:CURRent
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.current.
↪read()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13.4 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.deviation.
↪fetch()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:DEViation
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.deviation.
↪read()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13.5 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: float: Frequency of the measured AF signal Unit: Hz
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.maximum.
↪fetch()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:AFSignal:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.voip.afSignal.maximum.
↪read()
```

Query the AF frequency and level results for the VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13.6 Frequency

class FrequencyCls

Frequency commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.voip.frequency.clone()
```

Subgroups

6.1.1.3.13.7 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.voip.frequency.delta.clone()
```

Subgroups

6.1.1.3.13.8 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:VOIP:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.average.
↪fetch()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:AVERage
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.average.
↪read()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.13.9 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:VOIP:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.current.
↪fetch()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:CURRent
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.current.
↪read()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.13.10 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:VOIP:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.deviation.
↪fetch()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>
↪:MEValuation:VOIP:FREQuency:DELTa:DEViation
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.deviation.
↪read()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.13.11 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>
↪:MEValuation:VOIP:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.maximum.
↪fetch()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:FREQuency:DELTa:MAXimum
value: float = driver.afRf.measurement.multiEval.voip.frequency.delta.maximum.
↪read()
```

Query delta results for AF frequency of VoIP path.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
frequency: Unit: Hz

6.1.1.3.13.12 Level

class LevelCls

Level commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.multiEval.voip.level.clone()
```

Subgroups

6.1.1.3.13.13 Delta

class DeltaCls

Delta 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.afRf.measurement.multiEval.voip.level.delta.clone()
```

Subgroups

6.1.1.3.13.14 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:AVERage
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:AVERage
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.average.
↪fetch()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:AVERage
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.average.
↪read()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13.15 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:CURRent
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:CURRent
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.current.
↪fetch()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:CURRent
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.current.
↪read()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13.16 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:DEViation
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:DEViation
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.
↪deviation.fetch()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:DEViation
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.
↪deviation.read()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.3.13.17 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:MAXimum
READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Level_Peak: float: Delta peak level of the AC component of the measured AF signal. Unit: %
- Level_Rms: float: Effective (RMS-averaged) delta level of the AC component of the measured AF signal. Unit: %

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.maximum.
↪fetch()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:MEValuation:VOIP:LEVel:DELTa:MAXimum
value: ResultData = driver.afRf.measurement.multiEval.voip.level.delta.maximum.
↪read()
```

Query delta results for AF level of VoIP path.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.4 SearchRoutines

SCPI Command:

```
INITiate:AFRF:MEASurement<Instance>:SROutines
STOP:AFRF:MEASurement<Instance>:SROutines
ABORt:AFRF:MEASurement<Instance>:SROutines
```

class SearchRoutinesCls

SearchRoutines commands group definition. 82 total commands, 6 Subgroups, 3 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:AFRF:MEASurement<Instance>:SROutines
driver.afRf.measurement.searchRoutines.abort()
```

Stops the search routine.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

initiate(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INITiate:AFRF:MEASurement<Instance>:SROutines
driver.afRf.measurement.searchRoutines.initiate()
```

Starts or continues the search routine.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

stop() → None

```
# SCPI: STOP:AFRF:MEASurement<Instance>:SROutines
driver.afRf.measurement.searchRoutines.stop()
```

Pauses the search routine.

stop_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: STOP:AFRF:MEASurement<Instance>:SROutines
driver.afRf.measurement.searchRoutines.stop_with_opc()
```

Pauses the search routine.

Same as stop, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.clone()
```

Subgroups

6.1.1.4.1 RifBandwidth

class RifBandwidthCls

RifBandwidth commands group definition. 16 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.rifBandwidth.clone()
```

Subgroups

6.1.1.4.1.1 Bandwidth

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:BANDwidth
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:BANDwidth
```

class BandwidthCls

Bandwidth commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:BANDwidth
value: float = driver.afRf.measurement.searchRoutines.rifBandwidth.bandwidth.
↪fetch()
```

Query the bandwidth as difference between higher frequency and lower frequency, that is the 'RX Bandwidth'.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bandwidth: 'RX Bandwidth' Range: 0 Hz to 1 MHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:BANDwidth
value: float = driver.afRf.measurement.searchRoutines.rifBandwidth.bandwidth.
↪read()
```

Query the bandwidth as difference between higher frequency and lower frequency, that is the 'RX Bandwidth'.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
bandwidth: 'RX Bandwidth' Range: 0 Hz to 1 MHz, Unit: Hz

6.1.1.4.1.2 Coffset

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:COFFset
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:COFFset
```

class CoffsetCls

Coffset commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:COFFset
value: float = driver.afRf.measurement.searchRoutines.rifBandwidth.coffset.
↪fetch()
```

Query the center frequency offset.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
center_offset: Center frequency offset Range: -100 kHz to 100 kHz, Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:COFFset
value: float = driver.afRf.measurement.searchRoutines.rifBandwidth.coffset.
↪read()
```

Query the center frequency offset.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
center_offset: Center frequency offset Range: -100 kHz to 100 kHz, Unit: Hz

6.1.1.4.1.3 Frequency

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency
```

class FrequencyCls

Frequency commands group definition. 4 total commands, 1 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Lower_Freq: float: Lower RF frequency Unit: Hz
- Higher_Freq: float: Higher RF frequency Unit: Hz

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency
value: ResultData = driver.afRf.measurement.searchRoutines.rifBandwidth.
↪frequency.fetch()
```

Query the lower and higher RF frequencies left and right from the nominal frequency. At the frequency values, the noise has increased to the noise target value (noise level method) . Or, the SINAD audio signal quality has dropped down to the target value (TIA-603-D method) .

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency
value: ResultData = driver.afRf.measurement.searchRoutines.rifBandwidth.
↪frequency.read()
```

Query the lower and higher RF frequencies left and right from the nominal frequency. At the frequency values, the noise has increased to the noise target value (noise level method) . Or, the SINAD audio signal quality has dropped down to the target value (TIA-603-D method) .

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.rifBandwidth.frequency.clone()
```

Subgroups

6.1.1.4.1.4 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rifBandwidth.
↪frequency.trace.fetch()
```

Query the x-values of the points in the RX bandwidth result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
rf_level: Comma-separated list of RF frequency values Unit: Hz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:FREQuency:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rifBandwidth.
↪frequency.trace.read()
```

Query the x-values of the points in the RX bandwidth result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
rf_level: Comma-separated list of RF frequency values Unit: Hz

6.1.1.4.1.5 Nlevel

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel
```

class NlevelCls

Nlevel commands group definition. 4 total commands, 1 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Lower_Noise_Level: float: Noise level at lower frequency Unit: V
- Higher_Noise_Level: float: Noise level at higher frequency Unit: V

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel
value: ResultData = driver.afRf.measurement.searchRoutines.rifBandwidth.nlevel.
↪fetch()
```

Query the lower and higher noise level values for the RX IF bandwidth measurement.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel
value: ResultData = driver.afRf.measurement.searchRoutines.rifBandwidth.nlevel.
↪read()
```

Query the lower and higher noise level values for the RX IF bandwidth measurement.

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.rifBandwidth.nlevel.clone()
```

Subgroups

6.1.1.4.1.6 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rifBandwidth.nlevel.
↪trace.fetch()
```

Query the noise level values for the RX IF bandwidth measurement. These values are the y-values for the points in the RX IF bandwidth diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
noise_level: Comma-separated list of noise level values Unit: V

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:NLEVel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rifBandwidth.nlevel.
↪trace.read()
```

Query the noise level values for the RX IF bandwidth measurement. These values are the y-values for the points in the RX IF bandwidth diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
noise_level: Comma-separated list of noise level values Unit: V

6.1.1.4.1.7 SignalQuality

class SignalQualityCls

SignalQuality 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.afRf.measurement.searchRoutines.rifBandwidth.signalQuality.clone()
```

Subgroups

6.1.1.4.1.8 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SQUality:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SQUality:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SQUality:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rifBandwidth.
↪signalQuality.trace.fetch()
```

Query the y-values of the points in the RX bandwidth result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
signal_level: Comma-separated list of signal quality values Unit: dB

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SQUality:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rifBandwidth.
↪signalQuality.trace.read()
```

Query the y-values of the points in the RX bandwidth result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
signal_level: Comma-separated list of signal quality values Unit: dB

6.1.1.4.1.9 Slevel

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SLEVel
READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SLEVel
```

class SlevelCls

Slevel commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Lower_Signal_Level: float: Signal quality level at the lower frequency Unit: dBm
- Higher_Signal_Level: float: Signal quality level at the higher frequency Unit: dBm

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SLEVel
value: ResultData = driver.afRf.measurement.searchRoutines.rifBandwidth.slevel.
↪fetch()
```

Query the signal quality level at the lower and higher frequency.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SLEVel
value: ResultData = driver.afRf.measurement.searchRoutines.rifBandwidth.slevel.
↪read()
```

Query the signal quality level at the lower and higher frequency.

return
structure: for return value, see the help for ResultData structure arguments.

6.1.1.4.2 Rsensitivity

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity
CALCulate:AFRF:MEASurement<Instance>:SROutines:RSENsitivity
READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity
```

class RsensitivityCls

Rsensitivity commands group definition. 10 total commands, 3 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'

- Sensitivity_Level: enums.ResultStatus: Measured RX sensitivity level (RF level) Unit: dBm
- Signal_Quality: enums.ResultStatus: Audio signal quality value measured at the RX sensitivity level Unit: dB

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Sensitivity_Level: float: Measured RX sensitivity level (RF level) Unit: dBm
- Signal_Quality: float: Audio signal quality value measured at the RX sensitivity level Unit: dB

calculate() → CalculateStruct

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:RSENsitivity
value: CalculateStruct = driver.afRf.measurement.searchRoutines.rsensitivity.
↪calculate()
```

Query the single results of the RX sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity
value: ResultData = driver.afRf.measurement.searchRoutines.rsensitivity.fetch()
```

Query the single results of the RX sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity
value: ResultData = driver.afRf.measurement.searchRoutines.rsensitivity.read()
```

Query the single results of the RX sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.rsensitivity.clone()
```

Subgroups

6.1.1.4.2.1 RfLevel

class RfLevelCls

RfLevel 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.afRf.measurement.searchRoutines.rsensitivity.rfLevel.clone()
```

Subgroups

6.1.1.4.2.2 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:RFLevel:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:RFLevel:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:RFLevel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsensitivity.
↪rfLevel.trace.fetch()
```

Query the x-values of the points in the RX sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
sensitivity_level: Comma-separated list of RF levels Unit: dBm

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:RFLevel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsensitivity.
↪rfLevel.trace.read()
```

Query the x-values of the points in the RX sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
sensitivity_level: Comma-separated list of RF levels Unit: dBm

6.1.1.4.2.3 Sensitivity

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SENSitivity
CALCulate:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SENSitivity
READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SENSitivity
```

class SensitivityCls

Sensitivity commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → float

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SENSitivity
value: float = driver.afRf.measurement.searchRoutines.rsensitivity.sensitivity.
↪calculate()
```

Query the sensitivity level of the RX sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
rx_sensitivity_level: Measured RX sensitivity level (RF level) Unit: dBm

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SENSitivity
value: float = driver.afRf.measurement.searchRoutines.rsensitivity.sensitivity.
↪fetch()
```

Query the sensitivity level of the RX sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
rx_sensitivity_level: Measured RX sensitivity level (RF level) Unit: dBm

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SENSitivity
value: float = driver.afRf.measurement.searchRoutines.rsensitivity.sensitivity.
↪read()
```

Query the sensitivity level of the RX sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
rx_sensitivity_level: Measured RX sensitivity level (RF level) Unit: dBm

6.1.1.4.2.4 SignalQuality

class SignalQualityCls

SignalQuality 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.afRf.measurement.searchRoutines.rsensitivity.signalQuality.clone()
```

Subgroups

6.1.1.4.2.5 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SQUality:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SQUality:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SQUality:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsensitivity.
↪signalQuality.trace.fetch()
```

Query the y-values of the points in the RX sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
sens_sign_quality: Comma-separated list of signal quality values Unit: dB

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSENsitivity:SQUality:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsensitivity.
↪signalQuality.trace.read()
```

Query the y-values of the points in the RX sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
sens_sign_quality: Comma-separated list of signal quality values Unit: dB

6.1.1.4.3 Rsquelch

class RsquelchCls

Rsquelch commands group definition. 18 total commands, 8 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.rsquelch.clone()
```

Subgroups

6.1.1.4.3.1 Hysteresis

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:HYSTeresis
CALCulate:AFRF:MEASurement<Instance>:SROutines:RSQuelch:HYSTeresis
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:HYSTeresis
```

class HysteresisCls

Hysteresis commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → float

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:RSQuelch:HYSTeresis
value: float or bool = driver.afRf.measurement.searchRoutines.rsquelch.
↪hysteresis.calculate()
```

Query the difference between the squelch switch-off level and the squelch switch-on level, that is the squelch hysteresis. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
hysteresis: (float or boolean) Squelch hysteresis Range: 0 dB to 50 dB, Unit: dB

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:HYSTeresis
value: float = driver.afRf.measurement.searchRoutines.rsquelch.hysteresis.
↪fetch()
```

Query the difference between the squelch switch-off level and the squelch switch-on level, that is the squelch hysteresis. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
hysteresis: Squelch hysteresis Range: 0 dB to 50 dB, Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:HYSTeresis
value: float = driver.afRf.measurement.searchRoutines.rsquelch.hysteresis.read()
```

Query the difference between the squelch switch-off level and the squelch switch-on level, that is the squelch hysteresis. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
hysteresis: Squelch hysteresis Range: 0 dB to 50 dB, Unit: dB

6.1.1.4.3.2 Llist

class LlistCls

Llist 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.afRf.measurement.searchRoutines.rsquelch.llist.clone()
```

Subgroups

6.1.1.4.3.3 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:LLISt:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:LLISt:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:LLISt:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsquelch.llist.
↪trace.fetch()
```

Query the list of RF levels for the squelch measurement. These levels are the x-values for the points in the RX squelch diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
level_list: Comma-separated list of RF level values Unit: dBm

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:LLISt:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsquelch.llist.
↪trace.read()
```

Query the list of RF levels for the squelch measurement. These levels are the x-values for the points in the RX squelch diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
level_list: Comma-separated list of RF level values Unit: dBm

6.1.1.4.3.4 OfLevel

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFLevel
CALCulate:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFLevel
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFLevel
```

class OfLevelCls

OfLevel commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → float

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFLevel
value: float or bool = driver.afRf.measurement.searchRoutines.rsquelch.ofLevel.
↪calculate()
```

Query the RF level at which the DUT closes the squelch so that the audio signal is not muted anymore. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
off_level: (float or boolean) RF level at squelch switch-off level Range: -158 dBm to 16 dBm, Unit: dBm

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFLevel
value: float = driver.afRf.measurement.searchRoutines.rsquelch.ofLevel.fetch()
```

Query the RF level at which the DUT closes the squelch so that the audio signal is not muted anymore. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
off_level: RF level at squelch switch-off level Range: -158 dBm to 16 dBm, Unit: dBm

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFLevel
value: float = driver.afRf.measurement.searchRoutines.rsquelch.ofLevel.read()
```

Query the RF level at which the DUT closes the squelch so that the audio signal is not muted anymore. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
off_level: RF level at squelch switch-off level Range: -158 dBm to 16 dBm, Unit: dBm

6.1.1.4.3.5 OfsQuality

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFSQuality
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFSQuality
```

class OfsQualityCls

OfsQuality commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFSQuality
value: float = driver.afRf.measurement.searchRoutines.rsquelch.ofsQuality.
↪fetch()
```

Query the signal quality at the squelch switch-off level.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
off_sig_quality: Signal quality value at squelch switch-off level Range: -150 dB to 150 dB, Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:OFSQuality
value: float = driver.afRf.measurement.searchRoutines.rsquelch.ofsQuality.read()
```

Query the signal quality at the squelch switch-off level.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
off_sig_quality: Signal quality value at squelch switch-off level Range: -150 dB to 150 dB, Unit: dB

6.1.1.4.3.6 OnLevel

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONLevel
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONLevel
```

class OnLevelCls

OnLevel commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONLevel
value: float = driver.afRf.measurement.searchRoutines.rsquelch.onLevel.fetch()
```

Query the RF level at which the DUT switches on the squelch so that the audio signal is muted.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
on_level: RF level at squelch switch-on level Range: -158 dBm to 16 dBm, Unit: dBm

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONLevel
value: float = driver.afRf.measurement.searchRoutines.rsquelch.onLevel.read()
```

Query the RF level at which the DUT switches on the squelch so that the audio signal is muted.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
on_level: RF level at squelch switch-on level Range: -158 dBm to 16 dBm, Unit: dBm

6.1.1.4.3.7 OnsQuality

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONSQuality
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONSQuality
```

class OnsQualityCls

OnsQuality commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONSQuality
value: float = driver.afRf.measurement.searchRoutines.rsquelch.onsQuality.
↪fetch()
```

Query the signal quality at the squelch switch-on level.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
on_sig_quality: Signal quality at squelch switch-on level Range: -150 dB to 150 dB, Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:ONSQuality
value: float = driver.afRf.measurement.searchRoutines.rsquelch.onsQuality.read()
```

Query the signal quality at the squelch switch-on level.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
on_sig_quality: Signal quality at squelch switch-on level Range: -150 dB to 150 dB, Unit: dB

6.1.1.4.3.8 SignalQuality

class SignalQualityCls

SignalQuality 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.afRf.measurement.searchRoutines.rsquelch.signalQuality.clone()
```

Subgroups

6.1.1.4.3.9 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:SQUality:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:SQUality:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:SQUality:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsquelch.
↪signalQuality.trace.fetch()
```

Query the list of signal quality values for the squelch measurement. These values are the y-values for the points in the RX squelch diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
sig_qual_list: Comma-separated list of signal quality values Unit: dB

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:SQUality:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.rsquelch.
↪signalQuality.trace.read()
```

Query the list of signal quality values for the squelch measurement. These values are the y-values for the points in the RX squelch diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
sig_qual_list: Comma-separated list of signal quality values Unit: dB

6.1.1.4.3.10 Tlevel

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:TLEVel
READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:TLEVel
```

class TlevelCls

Tlevel commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:RSQuelch:TLEVel
value: float = driver.afRf.measurement.searchRoutines.rsquelch.tlevel.fetch()
```

Query the RF level at which the DUT opens the squelch if the DUT has a squelch control and has adjusted it to the maximum squelch switch-off level.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tight_level: RF level at maximum squelch switch-off level Range: -158 dBm to 16 dBm, Unit: dBm

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:RSQuelch:TLEVel
value: float = driver.afRf.measurement.searchRoutines.rsquelch.tlevel.read()
```

Query the RF level at which the DUT opens the squelch if the DUT has a squelch control and has adjusted it to the maximum squelch switch-off level.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tight_level: RF level at maximum squelch switch-off level Range: -158 dBm to 16 dBm, Unit: dBm

6.1.1.4.4 Ssnr

class SsnrCls

Ssnr 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.afRf.measurement.searchRoutines.ssnr.clone()
```

Subgroups

6.1.1.4.4.1 Average

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:AVERage
READ:AFRF:MEASurement<Instance>:SROutines:SSNR:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:AVERage
value: float = driver.afRf.measurement.searchRoutines.ssnr.average.fetch()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:SSNR:AVERage
value: float = driver.afRf.measurement.searchRoutines.ssnr.average.read()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

6.1.1.4.4.2 Current

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:CURRent
READ:AFRF:MEASurement<Instance>:SROutines:SSNR:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:CURRent
value: float = driver.afRf.measurement.searchRoutines.ssnr.current.fetch()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:SSNR:CURRent
value: float = driver.afRf.measurement.searchRoutines.ssnr.current.read()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

6.1.1.4.4.3 Deviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:DEViation
READ:AFRF:MEASurement<Instance>:SROutines:SSNR:DEViation
```

class DeviationCls

Deviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:DEViation
value: float = driver.afRf.measurement.searchRoutines.ssnr.deviation.fetch()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:SSNR:DEViation
value: float = driver.afRf.measurement.searchRoutines.ssnr.deviation.read()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

6.1.1.4.4.4 Maximum

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:MAXimum
READ:AFRF:MEASurement<Instance>:SROutines:SSNR:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:SSNR:MAXimum
value: float = driver.afRf.measurement.searchRoutines.ssnr.maximum.fetch()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:SSNR:MAXimum
value: float = driver.afRf.measurement.searchRoutines.ssnr.maximum.read()
```

Query the SNR results for the 'Switched SNR' search routine. A statistical evaluation of the SNR values is returned.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
ssnr: Switched SNR value Unit: dB

6.1.1.4.5 State

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:STATe
value: enums.ResourceState = driver.afRf.measurement.searchRoutines.state.
↪fetch()
```

Queries the main search routine state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.state.clone()
```

Subgroups

6.1.1.4.5.1 All

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[ResourceState]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:STATe:ALL
value: List[enums.ResourceState] = driver.afRf.measurement.searchRoutines.state.
↪all.fetch()
```

Queries the main search routine state and all substates. The substates provide additional information for the main state RUN.

return

meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

6.1.1.4.6 Tsensitivity

class TsensitivityCls

Tsensitivity commands group definition. 25 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.clone()
```

Subgroups

6.1.1.4.6.1 AudioOutput

class AudioOutputCls

AudioOutput commands group definition. 5 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.audioOutput.clone()
```

Subgroups

6.1.1.4.6.2 Level

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel
CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel
```

class LevelCls

Level commands group definition. 5 total commands, 1 Subgroups, 3 group commands

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel
value: enums.ResultStatus = driver.afRf.measurement.searchRoutines.tsensitivity.
↪audioOutput.level.calculate()
```

Query the AF signal level result at the target deviation for the AF path for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tx_mod_sens: Level of the AF signal for the AF path Unit: V

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.audioOutput.
↪level.fetch()
```

Query the AF signal level result at the target deviation for the AF path for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tx_mod_sens: Level of the AF signal for the AF path Unit: V

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.audioOutput.
↪level.read()
```

Query the AF signal level result at the target deviation for the AF path for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tx_mod_sens: Level of the AF signal for the AF path Unit: V

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.audioOutput.level.clone()
```

Subgroups

6.1.1.4.6.3 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪audioOutput.level.trace.fetch()
```

Query the x-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
af_level_list: Comma-separated list of AF level values for the AF path Unit: V

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:AOUT:LEVel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪audioOutput.level.trace.read()
```

Query the x-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
af_level_list: Comma-separated list of AF level values for the AF path Unit: V

6.1.1.4.6.4 Fdeviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation
CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation
```

class FdeviationCls

Fdeviation commands group definition. 5 total commands, 1 Subgroups, 3 group commands

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation
value: enums.ResultStatus = driver.afRf.measurement.searchRoutines.tsensitivity.
↪fdeviation.calculate()
```

Query the frequency deviation result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
freq_dev: Frequency deviation of the demodulated signal Unit: Hz

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.fdeviation.
↪fetch()
```

Query the frequency deviation result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
freq_dev: Frequency deviation of the demodulated signal Unit: Hz

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.fdeviation.
↪read()
```

Query the frequency deviation result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
freq_dev: Frequency deviation of the demodulated signal Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.fdeviation.clone()
```

Subgroups

6.1.1.4.6.5 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪fdeviation.trace.fetch()
```

Query the y-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
freq_dev_list: Comma-separated list of frequency deviation values Unit: Hz

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:FDEViation:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪fdeviation.trace.read()
```

Query the y-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
freq_dev_list: Comma-separated list of frequency deviation values Unit: Hz

6.1.1.4.6.6 ModDepth

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth
CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth
```

class ModDepthCls

ModDepth commands group definition. 5 total commands, 1 Subgroups, 3 group commands

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth
value: enums.ResultStatus = driver.afRf.measurement.searchRoutines.tsensitivity.
↪modDepth.calculate()
```

Query the modulation depth result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
mod_depth: Modulation depth of the demodulated signal Unit: %

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.modDepth.
↪fetch()
```

Query the modulation depth result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
mod_depth: Modulation depth of the demodulated signal Unit: %

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.modDepth.
↪read()
```

Query the modulation depth result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
mod_depth: Modulation depth of the demodulated signal Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.modDepth.clone()
```

Subgroups

6.1.1.4.6.7 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪modDepth.trace.fetch()
```

Query the y-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
mod_depth_list: Comma-separated list of modulation depth values Unit: %

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:MDEPth:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪modDepth.trace.read()
```

Query the y-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
mod_depth_list: Comma-separated list of modulation depth values Unit: %

6.1.1.4.6.8 Pdeviation

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation
CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation
```

class PdeviationCls

Pdeviation commands group definition. 5 total commands, 1 Subgroups, 3 group commands

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation
value: enums.ResultStatus = driver.afRf.measurement.searchRoutines.tsensitivity.
↪pdeviation.calculate()
```

Query the phase deviation result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
phase_dev: Phase deviation of the demodulated signal Unit: deg

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.pdeviation.
↪fetch()
```

Query the phase deviation result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
phase_dev: Phase deviation of the demodulated signal Unit: deg

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.pdeviation.
↪read()
```

Query the phase deviation result for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
phase_dev: Phase deviation of the demodulated signal Unit: deg

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.pdeviation.clone()
```

Subgroups

6.1.1.4.6.9 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪pdeviation.trace.fetch()
```

Query the y-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
phase_dev_list: Comma-separated list of phase deviation values Unit: deg

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:PDEViation:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.
↪pdeviation.trace.read()
```

Query the y-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
phase_dev_list: Comma-separated list of phase deviation values Unit: deg

6.1.1.4.6.10 Voip

class VoipCls

Voip commands group definition. 5 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.voip.clone()
```

Subgroups

6.1.1.4.6.11 Level

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel
CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel
```

class LevelCls

Level commands group definition. 5 total commands, 1 Subgroups, 3 group commands

calculate() → ResultStatus

```
# SCPI: CALCulate:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel
value: enums.ResultStatus = driver.afRf.measurement.searchRoutines.tsensitivity.
↪voip.level.calculate()
```

Query the AF signal level result at the target deviation in the VoIP path for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tx_mod_sens: Level of the AF signal for the VoIP path Unit: dBm0

fetch() → float

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.voip.level.
↪fetch()
```

Query the AF signal level result at the target deviation in the VoIP path for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tx_mod_sens: Level of the AF signal for the VoIP path Unit: dBm0

read() → float

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel
value: float = driver.afRf.measurement.searchRoutines.tsensitivity.voip.level.
↪read()
```

Query the AF signal level result at the target deviation in the VoIP path for the TX modulation sensitivity search routine. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
tx_mod_sens: Level of the AF signal for the VoIP path Unit: dBm0

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.afRf.measurement.searchRoutines.tsensitivity.voip.level.clone()
```

Subgroups

6.1.1.4.6.12 Trace

SCPI Command:

```
FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel:TRACe
READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel:TRACe
```

class TraceCls

Trace commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.voip.
↪level.trace.fetch()
```

Query the x-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
af_level_list: Comma-separated list of AF levels of the VoIP path Unit: dBm0

read() → List[float]

```
# SCPI: READ:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:VOIP:LEVel:TRACe
value: List[float] = driver.afRf.measurement.searchRoutines.tsensitivity.voip.
↪level.trace.read()
```

Query the x-values of the points in the TX modulation sensitivity result diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
af_level_list: Comma-separated list of AF levels of the VoIP path Unit: dBm0

6.2 Calibration

class CalibrationCls

Calibration commands group definition. 7 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calibration.clone()
```

Subgroups

6.2.1 Base

SCPI Command:

```
CALibration:BASE:ALL
CALibration:BASE:ACFile
```

class BaseCls

Base commands group definition. 4 total commands, 1 Subgroups, 2 group commands

class AcFileStruct

Structure for reading output parameters. Fields:

- Type_Py: str: Name of the correction file as string
- Date: str: Creation date as string

class AllStruct

Structure for reading output parameters. Fields:

- Date: List[str]: Date of the calibration as string
- Time: List[str]: Time of the calibration as string
- Type_Py: List[enums.CalibType]: FSCorrection | CALibration | OGCal Type of the calibration FS-Correction Correction performed in factory or service CALibration Verification in the factory OGCal Verification by the service (outgoing calibration)

get_ac_file() → AcFileStruct

```
# SCPI: CALibration:BASE:ACFile
value: AcFileStruct = driver.calibration.base.get_ac_file()
```

Query name and creation date of the currently active RF path correction file. Possible result: 'Factory Default', '2016-09-28'

return
structure: for return value, see the help for AcFileStruct structure arguments.

get_all() → AllStruct

```
# SCPI: CALibration:BASE:ALL
value: AllStruct = driver.calibration.base.get_all()
```

Queries the stored calibration information. A comma-separated list is returned, containing three parameters per calibration, as described below.

return
structure: for return value, see the help for AllStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calibration.base.clone()
```

Subgroups

6.2.1.1 Latest

SCPI Command:

```
CALibration:BASE:LATest
```

class LatestCls

Latest commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Date: str: Date of the calibration as string
- Time: str: Time of the calibration as string
- Type_Py: enums.CalibType: FSCorrection | CALibration | OGCal Type of the calibration FSCorrection Correction performed in factory or service CALibration Verification in the factory OGCal Verification by the service (outgoing calibration)

get(*type_py: CalibType = None*) → GetStruct

```
# SCPI: CALibration:BASE:LATest
value: GetStruct = driver.calibration.base.latest.get(type_py = enums.CalibType.
↪CALibration)
```

Queries information about the latest calibration. Optionally, specify <Type> to query information about the latest calibration of a specific type. The information is returned as <Date>,<Time>,<Type>.

param type_py
FSCorrection | CALibration | OGCal Type of the calibration FSCorrection Correction performed in factory or service CALibration Verification in the factory OGCal Verification by the service (outgoing calibration)

return
structure: for return value, see the help for GetStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.calibration.base.latest.clone()
```

Subgroups

6.2.1.1.1 Specific

SCPI Command:

```
CALibration:BASE:LATest:SPECific
```

class SpecificCls

Specific commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Date: str: Date of the calibration as string
- Time: str: Time of the calibration as string

get(*mode: CalibType*) → GetStruct

```
# SCPI: CALibration:BASE:LATest:SPECific
value: GetStruct = driver.calibration.base.latest.specific.get(mode = enums.
↪CalibType.CALibration)
```

Queries date and time of the latest calibration of the specified type.

param mode
FSCorrection | CALibration | OGCal Type of the calibration for which information is queried FSCorrection Correction performed in factory or service CALibration Verification in the factory OGCal Verification by the service (outgoing calibration)

return
structure: for return value, see the help for GetStruct structure arguments.

6.2.2 GprfMeasurement

class GprfMeasurementCls

GprfMeasurement 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.calibration.gprfMeasurement.clone()
```

Subgroups

6.2.2.1 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.gprfMeasurement.extPwrSensor.clone()
```

Subgroups

6.2.2.1.1 Zero

SCPI Command:

```
CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
```

class ZeroCls

Zero commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → Status

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
value: enums.Status = driver.calibration.gprfMeasurement.extPwrSensor.zero.get()
```

Initiates zeroing of the power sensor. A query returns whether the zeroing was successful.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
zeroing_state: PASSed | FAILed

set() → None

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:EPSensor:ZERO
driver.calibration.gprfMeasurement.extPwrSensor.zero.set()
```

Initiates zeroing of the power sensor. A query returns whether the zeroing was successful.

Use RsCma.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.gprfMeasurement.extPwrSensor.zero.set_with_opc()
```

Initiates zeroing of the power sensor. A query returns whether the zeroing was successful.

Use RsCma.reliability.last_value to read the updated reliability indicator.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.2.2.2 Nrt

class NrtCls

Nrt 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.gprfMeasurement.nrt.clone()
```

Subgroups

6.2.2.2.1 Zero

SCPI Command:

```
CALibration:GPRF:MEASurement<Instance>:NRT:ZERO
```

class ZeroCls

Zero commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → Status

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:NRT:ZERO
value: enums.Status = driver.calibration.gprfMeasurement.nrt.zero.get()
```

Initiates zeroing of the power sensor. A query returns whether the zeroing was successful.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
zeroing_state: PASSed | FAILed

set() → None

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:NRT:ZERO
driver.calibration.gprfMeasurement.nrt.zero.set()
```

Initiates zeroing of the power sensor. A query returns whether the zeroing was successful.

Use RsCma.reliability.last_value to read the updated reliability indicator.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:NRT:ZERO
driver.calibration.gprfMeasurement.nrt.zero.set_with_opc()
```

Initiates zeroing of the power sensor. A query returns whether the zeroing was successful.

Use RsCma.reliability.last_value to read the updated reliability indicator.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.2.2.3 Spectrum

SCPI Command:

```
CALibration:GPRF:MEASurement<Instance>:SPECtrum:TGENerator
```

class SpectrumCls

Spectrum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tgenerator() → bool

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:SPECtrum:TGENerator
value: bool = driver.calibration.gprfMeasurement.spectrum.get_tgenerator()
```

Starts or aborts the calibration for a tracking generator setup.

return
calibrate: OFF | ON ON Setting ON starts the calibration. Return value ON indicates an ongoing calibration. OFF Setting OFF aborts an ongoing calibration. Return value OFF indicates that there is no ongoing calibration.

set_tgenerator(*calibrate: bool*) → None

```
# SCPI: CALibration:GPRF:MEASurement<Instance>:SPECtrum:TGENerator
driver.calibration.gprfMeasurement.spectrum.set_tgenerator(calibrate = False)
```

Starts or aborts the calibration for a tracking generator setup.

param calibrate
OFF | ON ON Setting ON starts the calibration. Return value ON indicates an ongoing calibration. OFF Setting OFF aborts an ongoing calibration. Return value OFF indicates that there is no ongoing calibration.

6.3 Configure

class ConfigureCls

Configure commands group definition. 622 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.clone()
```

Subgroups

6.3.1 AfRf

class AfRfCls

AfRf commands group definition. 402 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.clone()
```

Subgroups

6.3.1.1 Generator

class GeneratorCls

Generator commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.generator.clone()
```

Subgroups

6.3.1.1.1 Voip

class VoipCls

Voip 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.afRf.generator.voip.clone()
```

Subgroups

6.3.1.1.1.1 Frequency

SCPI Command:

```
CONFigure:AFRF:GENerator<Instance>:VOIP:FREQuency:ATMFrequency
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_atm_frequency() → bool

```
# SCPI: CONFigure:AFRF:GENerator<Instance>:VOIP:FREQuency:ATMFrequency
value: bool = driver.configure.afRf.generator.voip.frequency.get_atm_frequency()
```

No command help available

return
atm_frequency: No help available

set_atm_frequency(*atm_frequency: bool*) → None

```
# SCPI: CONFigure:AFRF:GENerator<Instance>:VOIP:FREQuency:ATMFrequency
driver.configure.afRf.generator.voip.frequency.set_atm_frequency(atm_frequency␣
↪= False)
```

No command help available

param atm_frequency
No help available

6.3.1.2 Measurement

class MeasurementCls

Measurement commands group definition. 401 total commands, 15 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.clone()
```

Subgroups

6.3.1.2.1 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.afRf.measurement.audioInput.repcap_audioInput_get()
driver.configure.afRf.measurement.audioInput.repcap_audioInput_set(repcap.AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 32 total commands, 13 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.clone()
```

Subgroups

6.3.1.2.1.1 Aranging

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ARANging
```

class ArangingCls

Aranging commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ARANging
value: bool = driver.configure.afRf.measurement.audioInput.aranging.
↪get(audioInput = repcap.AudioInput.Default)
```

Enables or disables auto ranging for an AF IN connector.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
enable: OFF | ON Switches auto ranging off or on

set(*enable: bool*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ARANging
driver.configure.afRf.measurement.audioInput.aranging.set(enable = False,␣
↪audioInput = repcap.AudioInput.Default)
```

Enables or disables auto ranging for an AF IN connector.

param enable
OFF | ON Switches auto ranging off or on

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.2 Counter

class CounterCls

Counter 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.afRf.measurement.audioInput.counter.clone()
```

Subgroups

6.3.1.2.1.3 Mode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:COUNter:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → FreqCounterMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:COUNter:MODE
value: enums.FreqCounterMode = driver.configure.afRf.measurement.audioInput.
↪counter.mode.get(audioInput = repcap.AudioInput.Default)
```

Selects the type of frequency counter for measuring the AF frequency.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
freq_counter_mode: SW | HW Software or hardware implementation of the frequency counter

set(*freq_counter_mode: FreqCounterMode*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:COUNter:MODE
driver.configure.afRf.measurement.audioInput.counter.mode.set(freq_counter_mode␣
↪= enums.FreqCounterMode.HW, audioInput = repcap.AudioInput.Default)
```

Selects the type of frequency counter for measuring the AF frequency.

param freq_counter_mode
SW | HW Software or hardware implementation of the frequency counter

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.4 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ENABle
value: bool = driver.configure.afRf.measurement.audioInput.enable.
↪get(audioInput = repcap.AudioInput.Default)
```

Enables or disables an AF IN connector.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
test_af: OFF | ON Switches the connector off or on

set(*test_af: bool*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ENABle
driver.configure.afRf.measurement.audioInput.enable.set(test_af = False,␣
↪audioInput = repcap.AudioInput.Default)
```

Enables or disables an AF IN connector.

param test_af
OFF | ON Switches the connector off or on

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.5 FilterPy

class FilterPyCls

FilterPy commands group definition. 12 total commands, 8 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.filterPy.clone()
```

Subgroups

6.3.1.2.1.6 Bpass

class BpassCls

Bpass commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.filterPy.bpass.clone()
```

Subgroups

6.3.1.2.1.7 Bandwidth

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:BWIDth
```

class BandwidthCls

Bandwidth commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:BWIDth
value: float = driver.configure.afRf.measurement.audioInput.filterPy.bpass.
↪bandwidth.get(audioInput = repcap.AudioInput.Default)
```

Configures the bandwidth of the variable bandpass filter in an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
bandwidth: Range: 20 Hz to 20 kHz, Unit: Hz

set(*bandwidth: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:BWIDth
driver.configure.afRf.measurement.audioInput.filterPy.bpass.bandwidth.
↪set(bandwidth = 1.0, audioInput = repcap.AudioInput.Default)
```

Configures the bandwidth of the variable bandpass filter in an AF input path.

param bandwidth
Range: 20 Hz to 20 kHz, Unit: Hz

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.8 Cfrequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:CFRequency
```

class CfrequencyCls

Cfrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:CFRequency
value: float = driver.configure.afRf.measurement.audioInput.filterPy.bpass.
↪cfrequency.get(audioInput = repcap.AudioInput.Default)
```

Configures the center frequency of the variable bandpass filter in an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
frequency: Range: 0 Hz to 21 kHz, Unit: Hz

set(*frequency: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:CFRequency
driver.configure.afRf.measurement.audioInput.filterPy.bpass.cfrequency.
↪set(frequency = 1.0, audioInput = repcap.AudioInput.Default)
```

Configures the center frequency of the variable bandpass filter in an AF input path.

param frequency
Range: 0 Hz to 21 kHz, Unit: Hz

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.9 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:ENABle
value: bool = driver.configure.afRf.measurement.audioInput.filterPy.bpass.
↪enable.get(audioInput = repcap.AudioInput.Default)
```

Enables or disables the variable bandpass filter in an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
enable: OFF | ON

set(*enable: bool*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:BPASs:ENABle
driver.configure.afRf.measurement.audioInput.filterPy.bpass.enable.set(enable =␣
↪False, audioInput = repcap.AudioInput.Default)
```

Enables or disables the variable bandpass filter in an AF input path.

param enable
OFF | ON

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.10 Dfrequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DFRequency
```

class DfrequencyCls

Dfrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DFRequency
value: float = driver.configure.afRf.measurement.audioInput.filterPy.dfrequency.
↪get(audioInput = repcap.AudioInput.Default)
```

Configures the reference frequency for single-tone measurements via an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
distor_freq: Range: 1 Hz to 10.5 kHz, Unit: Hz

set(*distor_freq: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DFRequency
driver.configure.afRf.measurement.audioInput.filterPy.dfrequency.set(distor_
↪freq = 1.0, audioInput = repcap.AudioInput.Default)
```

Configures the reference frequency for single-tone measurements via an AF input path.

param distor_freq
Range: 1 Hz to 10.5 kHz, Unit: Hz

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.11 Dwidth

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DWIDth
```

class DwidthCls

Dwidth commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class DwidthStruct

Response structure. Fields:

- Dwidth: enums.PwrFilterType: WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth
- Relative: enums.Relative: RELative | CONStant Bandwidth proportional to reference frequency or constant

get(*audioInput=AudioInput.Default*) → DwidthStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DWIDth
value: DwidthStruct = driver.configure.afRf.measurement.audioInput.filterPy.
↪dwidth.get(audioInput = repcap.AudioInput.Default)
```

Configures the bandwidth of the distortion filter in an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for DwidthStruct structure arguments.

set(*dwidth: PwrFilterType*, *relative: Relative*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DWIDth
driver.configure.afRf.measurement.audioInput.filterPy.dwidth.set(dwidth = enums.
↪PwrFilterType.NARRow, relative = enums.Relative.CONStant, audioInput = repcap.
↪AudioInput.Default)
```

Configures the bandwidth of the distortion filter in an AF input path.

param dwidth
WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth

param relative
RELative | CONStant Bandwidth proportional to reference frequency or constant

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.filterPy.dwidth.clone()
```

Subgroups

6.3.1.2.1.12 Sfactor

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DWIDth:SFACtor
```

class SfactorCls

Sfactor commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DWIDth:SFACtor
value: float = driver.configure.afRf.measurement.audioInput.filterPy.dwidth.
↪sfactor.get(audioInput = repcap.AudioInput.Default)
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:AIN1:FILT:DWID UDEF

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
factor: Range: 0.001 to 0.005

set(*factor: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:DWIDth:SFACtor
driver.configure.afRf.measurement.audioInput.filterPy.dwidth.sfactor.set(factor␣
↪= 1.0, audioInput = repcap.AudioInput.Default)
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:AIN1:FILT:DWID UDEF

param factor
Range: 0.001 to 0.005

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.13 Hpass

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:HPASs
```

class HpassCls

Hpass commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → HighpassFilterExtended

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:HPASs
value: enums.HighpassFilterExtended = driver.configure.afRf.measurement.
↪audioInput.filterPy.hpass.get(audioInput = repcap.AudioInput.Default)
```

Configures the highpass filter in an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
filter_py: OFF | F6 | F50 | F300 OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz

set(*filter_py: HighpassFilterExtended*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:HPASs
driver.configure.afRf.measurement.audioInput.filterPy.hpass.set(filter_py =␣
↪enums.HighpassFilterExtended.F300, audioInput = repcap.AudioInput.Default)
```

Configures the highpass filter in an AF input path.

param filter_py
OFF | F6 | F50 | F300 OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.14 Lpass

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:LPASs
```

class LpassCls

Lpass commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → LowpassFilterExtended

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:LPASs
value: enums.LowpassFilterExtended = driver.configure.afRf.measurement.
↪audioInput.filterPy.lpass.get(audioInput = repcap.AudioInput.Default)
```

Configures the lowpass filter in an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
filter_py: OFF | F255 | F3K | F3K4 | F4K | F15K OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

set(*filter_py: LowpassFilterExtended*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:LPASs
driver.configure.afRf.measurement.audioInput.filterPy.lpass.set(filter_py =␣
↪enums.LowpassFilterExtended.F15K, audioInput = repcap.AudioInput.Default)
```

Configures the lowpass filter in an AF input path.

param filter_py
OFF | F255 | F3K | F3K4 | F4K | F15K OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.15 Notch<Notch>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.afRf.measurement.audioInput.filterPy.notch.repcap_notch_get()
driver.configure.afRf.measurement.audioInput.filterPy.notch.repcap_notch_set(repcap.
↪Notch.Nr1)
```

class NotchCls

Notch commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Notch, default value after init: Notch.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.filterPy.notch.clone()
```

Subgroups

6.3.1.2.1.16 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:NOTCh<Num>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*, *notch=Notch.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:NOTCh<Num>:ENABle
value: bool = driver.configure.afRf.measurement.audioInput.filterPy.notch.
↪enable.get(audioInput = repcap.AudioInput.Default, notch = repcap.Notch.
↪Default)
```

Enables the notch filters 1, 2 or 3 of the AF1 IN or AF2 IN connectors.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
filter_enable: OFF | ON

set(*filter_enable: bool*, *audioInput=AudioInput.Default*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:NOTCh<Num>:ENABle
driver.configure.afRf.measurement.audioInput.filterPy.notch.enable.set(filter_
↪enable = False, audioInput = repcap.AudioInput.Default, notch = repcap.Notch.
↪Default)
```

Enables the notch filters 1, 2 or 3 of the AF1 IN or AF2 IN connectors.

param filter_enable
OFF | ON

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.1.17 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:NOTCh<Num>:FREQuency
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*, *notch=Notch.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:NOTCh<Num>:FREQuency
value: float = driver.configure.afRf.measurement.audioInput.filterPy.notch.
↪frequency.get(audioInput = repcap.AudioInput.Default, notch = repcap.Notch.
↪Default)
```

Sets the frequency for the notch filters 1, 2 or 3 of the AF1 IN or AF2 IN connectors.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
filter_frequency: Range: 5 Hz to 21000 Hz, Unit: Hz

set(*filter_frequency: float*, *audioInput=AudioInput.Default*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:NOTCh<Num>:FREQuency
driver.configure.afRf.measurement.audioInput.filterPy.notch.frequency.
↪set(filter_frequency = 1.0, audioInput = repcap.AudioInput.Default, notch =␣
↪repcap.Notch.Default)
```

Sets the frequency for the notch filters 1, 2 or 3 of the AF1 IN or AF2 IN connectors.

param filter_frequency
Range: 5 Hz to 21000 Hz, Unit: Hz

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.1.18 RobustAuto

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:ROBustauto
```

class RobustAutoCls

RobustAuto commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:ROBustauto
value: bool = driver.configure.afRf.measurement.audioInput.filterPy.robustAuto.
↪get(audioInput = repcap.AudioInput.Default)
```

Enables or disables robust automatic mode for distortion signal filtering in the AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
automatic_mode: OFF | ON

set(*automatic_mode: bool*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:ROBustauto
driver.configure.afRf.measurement.audioInput.filterPy.robustAuto.set(automatic_
↪mode = False, audioInput = repcap.AudioInput.Default)
```

Enables or disables robust automatic mode for distortion signal filtering in the AF input path.

param automatic_mode
OFF | ON

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.19 Weighting

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:WEIGhting
```

class WeightingCls

Weighting commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → WeightingFilter

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:WEIGhting
value: enums.WeightingFilter = driver.configure.afRf.measurement.audioInput.
↪filterPy.weighting.get(audioInput = repcap.AudioInput.Default)
```

Configures the weighting filter in an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
filter_py: OFF | AWEighting | CCITt | CMESsage OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter

set(*filter_py: WeightingFilter*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FILTer:WEIGhting
driver.configure.afRf.measurement.audioInput.filterPy.weighting.set(filter_py =␣
↪enums.WeightingFilter.AWEighting, audioInput = repcap.AudioInput.Default)
```

Configures the weighting filter in an AF input path.

param filter_py
OFF | AWEighting | CCITt | CMESsage OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.20 First

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:MLEVel
```

class FirstCls

First commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_mlevel() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:MLEVel
value: float = driver.configure.afRf.measurement.audioInput.first.get_mlevel()
```

Specifies the maximum expected level for the AF1 IN connector. This setting is only relevant, if auto ranging is disabled. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

return
max_level: Range: 10E-6 V to 43 V, Unit: V

set_mlevel(*max_level: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:MLEVel
driver.configure.afRf.measurement.audioInput.first.set_mlevel(max_level = 1.0)
```

Specifies the maximum expected level for the AF1 IN connector. This setting is only relevant, if auto ranging is disabled. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

param max_level
Range: 10E-6 V to 43 V, Unit: V

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.first.clone()
```

Subgroups

6.3.1.2.1.21 Level

class LevelCls

Level 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.afRf.measurement.audioInput.first.level.clone()
```

Subgroups

6.3.1.2.1.22 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:LEVel:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:LEVel:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:LEVel:DELTa:MEASured
value: float = driver.configure.afRf.measurement.audioInput.first.level.delta.
↪get_measured()
```

Configures the AF1 level measured reference value.

return
meas_val: Unit: V

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:LEVel:DELTa:USER
value: float = driver.configure.afRf.measurement.audioInput.first.level.delta.
↪get_user()
```

Configures the AF1 level reference mode.

return
user_val: Unit: V

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:FIRSt:LEVel:DELTa:USER
driver.configure.afRf.measurement.audioInput.first.level.delta.set_user(user_
↪val = 1.0)
```

Configures the AF1 level reference mode.

param user_val
Unit: V

6.3.1.2.1.23 Frequency

class FrequencyCls

Frequency commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.frequency.clone()
```

Subgroups

6.3.1.2.1.24 Delta

class DeltaCls

Delta commands group definition. 4 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.frequency.delta.clone()
```

Subgroups

6.3.1.2.1.25 Measured

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:MEASured
```

class MeasuredCls

Measured commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:MEASured
value: float = driver.configure.afRf.measurement.audioInput.frequency.delta.
↪measured.get(audioInput = repcap.AudioInput.Default)
```

Configures the AF frequency measured reference value.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
meas_val: Unit: Hz

6.3.1.2.1.26 Mode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.audioInput.frequency.
↪delta.mode.get(audioInput = repcap.AudioInput.Default)
```

Configures the AF frequency reference mode.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
mode: NONE | MEAS | USER

set(*mode: DeltaMode*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:MODE
driver.configure.afRf.measurement.audioInput.frequency.delta.mode.set(mode =␣
↪enums.DeltaMode.MEAS, audioInput = repcap.AudioInput.Default)
```

Configures the AF frequency reference mode.

param mode
NONE | MEAS | USER

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.27 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.audioInput.frequency.delta.update.
↪set(audioInput = repcap.AudioInput.Default)
```

Triggers the update of the AF frequency measurement reference value.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

set_with_opc(*audioInput=AudioInput.Default*, *opc_timeout_ms: int = -1*) → None

6.3.1.2.1.28 User

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:USER
```

class UserCls

User commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:USER
value: float = driver.configure.afRf.measurement.audioInput.frequency.delta.
↪user.get(audioInput = repcap.AudioInput.Default)
```

Configures the AF frequency user reference value.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
user_val: Unit: Hz

set(*user_val: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:FREQuency:DELTa:USER
driver.configure.afRf.measurement.audioInput.frequency.delta.user.set(user_val␣
↪= 1.0, audioInput = repcap.AudioInput.Default)
```

Configures the AF frequency user reference value.

param user_val
Unit: Hz

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.29 Gcoupling

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:GCOupling
```

class GcouplingCls

Gcoupling commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → GeneratorCoupling

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:GCOupling
value: enums.GeneratorCoupling = driver.configure.afRf.measurement.audioInput.
↪gcoupling.get(audioInput = repcap.AudioInput.Default)
```

Couples an AF IN connector to an internal signal generator.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
coupling: OFF | GEN1 | GEN2 | GEN3 | GEN4 OFF No coupling GENn Coupled to audio generator n

set(*coupling: GeneratorCoupling*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:GCOupling
driver.configure.afRf.measurement.audioInput.gcoupling.set(coupling = enums.
↪GeneratorCoupling.GEN1, audioInput = repcap.AudioInput.Default)
```

Couples an AF IN connector to an internal signal generator.

param coupling
OFF | GEN1 | GEN2 | GEN3 | GEN4 OFF No coupling GENn Coupled to audio generator n

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.30 Icoupling

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ICOupling
```

class IcouplingCls

Icoupling commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → PathCoupling

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ICOupling
value: enums.PathCoupling = driver.configure.afRf.measurement.audioInput.
↪icoupling.get(audioInput = repcap.AudioInput.Default)
```

Configures whether the DC signal component is blocked at an AF IN connector, or not.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
path_coupling: AC | DC AC DC component blocked, only AC component available DC AC and DC component available

set(*path_coupling: PathCoupling*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:ICOupling
driver.configure.afRf.measurement.audioInput.icoupling.set(path_coupling =␣
↪enums.PathCoupling.AC, audioInput = repcap.AudioInput.Default)
```

Configures whether the DC signal component is blocked at an AF IN connector, or not.

param path_coupling
AC | DC AC DC component blocked, only AC component available DC AC and DC component available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.31 Level

class LevelCls

Level 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.afRf.measurement.audioInput.level.clone()
```

Subgroups

6.3.1.2.1.32 Delta

class DeltaCls

Delta commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.level.delta.clone()
```

Subgroups

6.3.1.2.1.33 Mode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:LEVel:DELTa:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:LEVel:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.audioInput.level.
↪delta.mode.get(audioInput = repcap.AudioInput.Default)
```

Configures the AF level reference mode.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
mode: NONE | MEAS | USER

set(*mode: DeltaMode*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:LEVel:DELTa:MODE
driver.configure.afRf.measurement.audioInput.level.delta.mode.set(mode = enums.
↪DeltaMode.MEAS, audioInput = repcap.AudioInput.Default)
```

Configures the AF level reference mode.

param mode
NONE | MEAS | USER

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.34 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:LEVel:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:LEVel:DELTa:UPDate
driver.configure.afRf.measurement.audioInput.level.delta.update.set(audioInput␣
↪= repcap.AudioInput.Default)
```

Triggers the update of the AF level measurement reference value.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

set_with_opc(*audioInput=AudioInput.Default*, *opc_timeout_ms: int = -1*) → None

6.3.1.2.1.35 Mlevel

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:MLEVel
```

class MlevelCls

Mlevel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:MLEVel
value: float = driver.configure.afRf.measurement.audioInput.mlevel.
↪get(audioInput = repcap.AudioInput.Default)
```

Specifies the maximum expected level for an AF IN connector in voltage-related units (Table 'Units relevant for remote commands') . This setting is only relevant, if auto ranging is disabled. The command sets the same unit for both AF IN connectors. If you want to set different level units (e.g. dBm) or set level for both connectors independently, use method RsCma.Configure.AfRf.Measurement.AudioInput.First.mlevel for AF1 IN and method RsCma.Configure.AfRf.Measurement. AudioInput.Second.mlevel for AF2 IN.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
max_level: Range: 10E-6 V to 43 V, Unit: V

set(*max_level: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:MLEVel
driver.configure.afRf.measurement.audioInput.mlevel.set(max_level = 1.0,␣
↪audioInput = repcap.AudioInput.Default)
```

Specifies the maximum expected level for an AF IN connector in voltage-related units (Table 'Units relevant for remote commands') . This setting is only relevant, if auto ranging is disabled. The command sets the same unit for both AF IN connectors. If you want to set different level units (e.g. dBm) or set level for both connectors independently, use method RsCma.Configure.AfRf.Measurement.AudioInput.First.mlevel for AF1 IN and method RsCma.Configure.AfRf.Measurement. AudioInput.Second.mlevel for AF2 IN.

param max_level
Range: 10E-6 V to 43 V, Unit: V

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.36 Sdecay

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:SDECay
```

class SdecayCls

Sdecay commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → SlowDecay

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:SDECay
value: enums.SlowDecay = driver.configure.afRf.measurement.audioInput.sdecay.
↪get(audioInput = repcap.AudioInput.Default)
```

Requires 'Auto Range' > 'ON', see method RsCma.Configure.AfRf.Measurement.AudioInput.Aranging.set. Sets longer decay times of the auto ranging procedure implying longer decay times of a digital automatic gain control (AGC) . You can set multiples of the standard decay time of the digital AGC.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
slow_decay: OFF | X2 | X3 | X4 | X10 OFF Standard decay of the digital AGC. X2 | X3 | X4 | X10 Sets for longer decay times using multiples of standard decay time of the digital AGC.

set(*slow_decay: SlowDecay*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:SDECay
driver.configure.afRf.measurement.audioInput.sdecay.set(slow_decay = enums.
↪SlowDecay.OFF, audioInput = repcap.AudioInput.Default)
```

Requires 'Auto Range' > 'ON', see method RsCma.Configure.AfRf.Measurement.AudioInput.Aranging.set. Sets longer decay times of the auto ranging procedure implying longer decay times of a digital automatic gain control (AGC) . You can set multiples of the standard decay time of the digital AGC.

param slow_decay
OFF | X2 | X3 | X4 | X10 OFF Standard decay of the digital AGC. X2 | X3 | X4 | X10 Sets for longer decay times using multiples of standard decay time of the digital AGC.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.1.37 Second

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:MLEVel
```

class SecondCls

Second commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_mlevel() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:MLEVel
value: float = driver.configure.afRf.measurement.audioInput.second.get_mlevel()
```

Specifies the maximum expected level for the AF2 IN connector. This setting is only relevant, if auto ranging is disabled. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

return
max_level: Range: 10E-6 V to 43 V, Unit: V

set_mlevel(*max_level: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:MLEVel
driver.configure.afRf.measurement.audioInput.second.set_mlevel(max_level = 1.0)
```

Specifies the maximum expected level for the AF2 IN connector. This setting is only relevant, if auto ranging is disabled. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

param max_level
Range: 10E-6 V to 43 V, Unit: V

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioInput.second.clone()
```

Subgroups

6.3.1.2.1.38 Level

class LevelCls

Level 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.afRf.measurement.audioInput.second.level.clone()
```

Subgroups

6.3.1.2.1.39 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:LEVel:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:LEVel:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:LEVel:DELTa:MEASured
value: float = driver.configure.afRf.measurement.audioInput.second.level.delta.
↪get_measured()
```

Configures the AF2 level measured reference value.

return
meas_val: Unit: V

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:LEVel:DELTa:USER
value: float = driver.configure.afRf.measurement.audioInput.second.level.delta.
↪get_user()
```

Configures the AF2 level reference mode.

return
user_val: Unit: V

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN:SECond:LEVel:DELTa:USER
driver.configure.afRf.measurement.audioInput.second.level.delta.set_user(user_
↪val = 1.0)
```

Configures the AF2 level reference mode.

param user_val
Unit: V

6.3.1.2.1.40 Tmode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:TMODe
```

class TmodeCls

Tmode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → ToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:TMODe
value: enums.ToneMode = driver.configure.afRf.measurement.audioInput.tmode.
↪get(audioInput = repcap.AudioInput.Default)
```

No command help available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
tone_mode: No help available

set(*tone_mode: ToneMode*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:TMODe
driver.configure.afRf.measurement.audioInput.tmode.set(tone_mode = enums.
↪ToneMode.NOISe, audioInput = repcap.AudioInput.Default)
```

No command help available

param tone_mode
No help available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.2 AudioOutput<AudioOutput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.afRf.measurement.audioOutput.repcap_audioOutput_get()
driver.configure.afRf.measurement.audioOutput.repcap_audioOutput_set(repcap.AudioOutput.
↪Nr1)
```

class AudioOutputCls

AudioOutput commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: AudioOutput, default value after init: AudioOutput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.audioOutput.clone()
```

Subgroups

6.3.1.2.2.1 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioOutput=AudioOutput.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:ENABle
value: bool = driver.configure.afRf.measurement.audioOutput.enable.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Enables or disables an AF OUT connector.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
enable: OFF | ON Switches the connector off or on

set(*enable: bool*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:ENABle
driver.configure.afRf.measurement.audioOutput.enable.set(enable = False,␣
↪audioOutput = repcap.AudioOutput.Default)
```

Enables or disables an AF OUT connector.

param enable
OFF | ON Switches the connector off or on

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.1.2.2.2 First

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AOUT:FIRSt:LEVel
```

class FirstCls

First commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_level() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT:FIRSt:LEVel
value: float = driver.configure.afRf.measurement.audioOutput.first.get_level()
```

Specifies the output level for the AF1 OUT connector. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

return
level: Range: 10E-6 V to 5 V, Unit: V

set_level(*level: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT:FIRSt:LEVel
driver.configure.afRf.measurement.audioOutput.first.set_level(level = 1.0)
```

Specifies the output level for the AF1 OUT connector. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

param level
Range: 10E-6 V to 5 V, Unit: V

6.3.1.2.2.3 Level

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioOutput=AudioOutput.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:LEVel
value: float = driver.configure.afRf.measurement.audioOutput.level.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Specifies the output level for an AF OUT connector in voltage-related units (Table 'Units relevant for remote commands') . The command sets the same unit for both AF OUT connectors. If you want to set different level units (e.g. dBm) or set level for both connectors independently, use method RsCma.Configure.AfRf.Measurement.AudioOutput.First.level for AF1 OUT and method RsCma.Configure.AfRf.Measurement.AudioOutput.Second.level for AF2 OUT.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
level: Range: 10E-6 V to 5 V, Unit: V

set(*level: float*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:LEVel
driver.configure.afRf.measurement.audioOutput.level.set(level = 1.0,␣
↪audioOutput = repcap.AudioOutput.Default)
```

Specifies the output level for an AF OUT connector in voltage-related units (Table 'Units relevant for remote commands') . The command sets the same unit for both AF OUT connectors. If you want to set different level units (e.g. dBm) or set level for both connectors independently, use method RsCma.Configure.AfRf.Measurement.AudioOutput.First.level for AF1 OUT and method RsCma.Configure.AfRf.Measurement.AudioOutput.Second.level for AF2 OUT.

param level
Range: 10E-6 V to 5 V, Unit: V

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.1.2.2.4 Second

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AOUT:SECond:LEVel
```

class SecondCls

Second commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_level() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT:SECond:LEVel
value: float = driver.configure.afRf.measurement.audioOutput.second.get_level()
```

Specifies the output level for the AF2 OUT connector. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

return
level: Range: 10E-6 V to 5 V, Unit: V

set_level(*level: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT:SECond:LEVel
driver.configure.afRf.measurement.audioOutput.second.set_level(level = 1.0)
```

Specifies the output level for the AF2 OUT connector. Use this command, if you want to set different level units, e.g. dBm (Table 'Units relevant for remote commands') , or set the level for both connectors independently.

param level
Range: 10E-6 V to 5 V, Unit: V

6.3.1.2.2.5 Source

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:SOURce
```

class SourceCls

Source commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioOutput=AudioOutput.Default*) → AudioSource

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:SOURce
value: enums.AudioSource = driver.configure.afRf.measurement.audioOutput.source.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Sets the audio signal source for an AF OUT connector.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
source: NONE | DEM | DEML | DEMR | VOIP | UGEN DEM Demodulator output (FM, PM, ...) DEML Demodulator output, left channel (FM stereo) DEMR Demodulator output, right channel (FM stereo) VOIP Generated audio signal transported via LAN (VoIP) UGEN User-generated audio signal

set(*source: AudioSource*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:SOURce
driver.configure.afRf.measurement.audioOutput.source.set(source = enums.
↪AudioSource.DEM, audioOutput = repcap.AudioOutput.Default)
```

Sets the audio signal source for an AF OUT connector.

param source
NONE | DEM | DEML | DEMR | VOIP | UGEN DEM Demodulator output (FM, PM, ...) DEML Demodulator output, left channel (FM stereo) DEMR Demodulator output, right channel (FM stereo) VOIP Generated audio signal transported via LAN (VoIP) UGEN User-generated audio signal

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.1.2.3 Cdefinition

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RCHannel
CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RFRequency
CONFigure:AFRF:MEASurement<Instance>:CDEFinition:CSPace
CONFigure:AFRF:MEASurement<Instance>:CDEFinition
```

class CdefinitionCls

Cdefinition commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_cspace() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition:CSPace
value: float = driver.configure.afRf.measurement.cdefinition.get_cspace()
```

Defines the channel spacing, that is the center frequency difference of two adjacent channels. This setting is part of the channel definition.

return
channel_space: Range: 100 Hz to 4 MHz, Unit: Hz

get_rchannel() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RCHannel
value: int = driver.configure.afRf.measurement.cdefinition.get_rchannel()
```

Assigns a reference channel number to the reference frequency defined via method RsCma.Configure.AfRf.Measurement. Cdefinition.rfrequency. This setting is part of the channel definition.

return
reference_ch: Range: 0 Ch to 9999 Ch, Unit: Ch

get_rfrequency() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RFRequency
value: float = driver.configure.afRf.measurement.cdefinition.get_rfrequency()
```

Assigns a reference frequency to the reference channel number defined via method RsCma.Configure.AfRf.Measurement. Cdefinition.rchannel. This setting is part of the channel definition.

return
reference_freq: Range: 100 kHz to 3 GHz, Unit: Hz

get_value() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition
value: bool = driver.configure.afRf.measurement.cdefinition.get_value()
```

Activates or deactivates the channel definition.

return
reference_ch: OFF | ON

set_cspace(*channel_space: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition:CSPace
driver.configure.afRf.measurement.cdefinition.set_cspace(channel_space = 1.0)
```

Defines the channel spacing, that is the center frequency difference of two adjacent channels. This setting is part of the channel definition.

param channel_space
Range: 100 Hz to 4 MHz, Unit: Hz

set_rchannel(*reference_ch: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RCHannel
driver.configure.afRf.measurement.cdefinition.set_rchannel(reference_ch = 1)
```

Assigns a reference channel number to the reference frequency defined via method RsCma.Configure.AfRf.Measurement. Cdefinition.rfrequency. This setting is part of the channel definition.

param reference_ch
Range: 0 Ch to 9999 Ch, Unit: Ch

set_rfrequency(*reference_freq: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RFRequency
driver.configure.afRf.measurement.cdefinition.set_rfrequency(reference_freq = 1.
↪0)
```

Assigns a reference frequency to the reference channel number defined via method RsCma.Configure.AfRf.Measurement. Cdefinition.rchannel. This setting is part of the channel definition.

param reference_freq
Range: 100 kHz to 3 GHz, Unit: Hz

set_value(*reference_ch: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:CDEFinition
driver.configure.afRf.measurement.cdefinition.set_value(reference_ch = False)
```

Activates or deactivates the channel definition.

param reference_ch
OFF | ON

6.3.1.2.4 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DELTa:ENABle
```

class DeltaCls

Delta commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DELTa:ENABle
value: bool = driver.configure.afRf.measurement.delta.get_enable()
```

Enables or disables delta measurements and delta measurement results. If disabled, delta measurement result views are hidden. For delta measurements in 'Repetition Mode' > 'SingleShot', the results for 'Standard Deviation' are marked 'NCAP', since they refer only to one measurement value.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DELTa:ENABle
driver.configure.afRf.measurement.delta.set_enable(enable = False)
```

Enables or disables delta measurements and delta measurement results. If disabled, delta measurement result views are hidden. For delta measurements in 'Repetition Mode' > 'SingleShot', the results for 'Standard Deviation' are marked 'NCAP', since they refer only to one measurement value.

param enable
OFF | ON

6.3.1.2.5 Demodulation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation
```

class DemodulationCls

Demodulation commands group definition. 36 total commands, 8 Subgroups, 1 group commands

get_value() → Demodulation

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation
value: enums.Demodulation = driver.configure.afRf.measurement.demodulation.get_
↪value()
```

Selects the type of demodulation to be performed.

return
demodulation: FMSTereo | FM | AM | USB | LSB | PM FMSTereo FM stereo multiplex signal FM, PM, AM Frequency / phase / amplitude modulation USB, LSB Single sideband modulation, upper / lower sideband

set_value(*demodulation: Demodulation*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation
driver.configure.afRf.measurement.demodulation.set_value(demodulation = enums.
↪Demodulation.AM)
```

Selects the type of demodulation to be performed.

param demodulation
FMSTereo | FM | AM | USB | LSB | PM FMSTereo FM stereo multiplex signal FM, PM, AM Frequency / phase / amplitude modulation USB, LSB Single sideband modulation, upper / lower sideband

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.clone()
```

Subgroups

6.3.1.2.5.1 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Test_Left: bool: No parameter help available
- Test_Right: bool: No parameter help available

get() → EnableStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:ENABle
value: EnableStruct = driver.configure.afRf.measurement.demodulation.enable.
↪get()
```

No command help available

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*test_left: bool*, *test_right: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:ENABle
driver.configure.afRf.measurement.demodulation.enable.set(test_left = False,␣
↪test_right = False)
```

No command help available

param test_left
No help available

param test_right
No help available

6.3.1.2.5.2 Fdeviation

class FdeviationCls

Fdeviation commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.fdeviation.clone()
```

Subgroups

6.3.1.2.5.3 Peak

class PeakCls

Peak commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.fdeviation.peak.clone()
```

Subgroups

6.3.1.2.5.4 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:PEAK:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:PEAK:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:PEAK:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:PEAK:DELTa:MEASured
value: float = driver.configure.afRf.measurement.demodulation.fdeviation.peak.
↪delta.get_measured()
```

Configures the measured peak frequency deviation reference value for delta measurement.

return
meas_val: Unit: Hz

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:PEAK:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.demodulation.
↪fdeviation.peak.delta.get_mode()
```

Configures the reference mode of the peak frequency deviation value for delta measurement.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:PEAK:DELTa:USER
value: float = driver.configure.afRf.measurement.demodulation.fdeviation.peak.
↪delta.get_user()
```

Configures the user peak frequency deviation reference value for delta measurement.

return
user_val: Unit: Hz

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:PEAK:DELTa:MODE
driver.configure.afRf.measurement.demodulation.fdeviation.peak.delta.set_
↪mode(mode = enums.DeltaMode.MEAS)
```

Configures the reference mode of the peak frequency deviation value for delta measurement.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:PEAK:DELTa:USER
driver.configure.afRf.measurement.demodulation.fdeviation.peak.delta.set_
↪user(user_val = 1.0)
```

Configures the user peak frequency deviation reference value for delta measurement.

param user_val
Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.fdeviation.peak.delta.clone()
```

Subgroups

6.3.1.2.5.5 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:PEAK:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:PEAK:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.fdeviation.peak.delta.update.
↪set()
```

Triggers the update of the peak frequency deviation reference value for delta measurement.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:PEAK:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.fdeviation.peak.delta.update.set_
↪with_opc()
```

Triggers the update of the peak frequency deviation reference value for delta measurement.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.5.6 Rms

class RmsCls

Rms commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.fdeviation.rms.clone()
```

Subgroups

6.3.1.2.5.7 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:RMS:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:RMS:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:RMS:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:RMS:DELTa:MEASured
value: float = driver.configure.afRf.measurement.demodulation.fdeviation.rms.
↪delta.get_measured()
```

Configures the measured RMS frequency deviation reference value for delta measurement.

return
meas_val: Unit: Hz

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:RMS:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.demodulation.
↪fdeviation.rms.delta.get_mode()
```

Configures the reference mode of the RMS frequency deviation value for delta measurement.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:RMS:DELTa:USER
value: float = driver.configure.afRf.measurement.demodulation.fdeviation.rms.
↪delta.get_user()
```

Configures the user RMS frequency deviation reference value for delta measurement.

return
user_val: Unit: Hz

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:RMS:DELTa:MODE
driver.configure.afRf.measurement.demodulation.fdeviation.rms.delta.set_
↪mode(mode = enums.DeltaMode.MEAS)
```

Configures the reference mode of the RMS frequency deviation value for delta measurement.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:RMS:DELTa:USER
driver.configure.afRf.measurement.demodulation.fdeviation.rms.delta.set_
↪user(user_val = 1.0)
```

Configures the user RMS frequency deviation reference value for delta measurement.

param user_val
Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.fdeviation.rms.delta.clone()
```

Subgroups

6.3.1.2.5.8 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FDEViation:RMS:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:RMS:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.fdeviation.rms.delta.update.set()
```

Triggers the update of the RMS frequency deviation reference value for delta measurement.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FDEViation:RMS:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.fdeviation.rms.delta.update.set_
↪with_opc()
```

Triggers the update of the RMS frequency deviation reference value for delta measurement.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.5.9 FilterPy

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:ENABle
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:WEIGhting
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DEEMphasis
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:ROBustauto
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:LPASs
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:HPASs
```

class FilterPyCls

FilterPy commands group definition. 14 total commands, 4 Subgroups, 6 group commands

get_deemphasis() → PreDeEmphasis

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DEEMphasis
value: enums.PreDeEmphasis = driver.configure.afRf.measurement.demodulation.
↪filterPy.get_deemphasis()
```

Configures the de-emphasis filter in the RF input path.

return
deemphasis: OFF | T50 | T75 | T750 OFF Filter disabled T50, T75, T750 Time constant 50 us / 75 us / 750 us

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:ENABle
value: bool = driver.configure.afRf.measurement.demodulation.filterPy.get_
↪enable()
```

Selects whether the demodulation results are measured before or after the filters in the RF input path. For FM stereo, the demodulation results are always measured before the filters.

return
disable: No help available

get_hpass() → HighpassFilterExtended

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:HPASs
value: enums.HighpassFilterExtended = driver.configure.afRf.measurement.
↪demodulation.filterPy.get_hpass()
```

Configures the highpass filter in the RF input path.

return
highpass: OFF | F6 | F50 | F300 OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz

get_lpass() → LowpassFilterExtended

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:LPASs
value: enums.LowpassFilterExtended = driver.configure.afRf.measurement.
↪demodulation.filterPy.get_lpass()
```

Configures the lowpass filter in the RF input path.

return
lowpass: OFF | F255 | F3K | F3K4 | F4K | F15K OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

get_robust_auto() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:ROBustauto
value: bool = driver.configure.afRf.measurement.demodulation.filterPy.get_
↪robust_auto()
```

Enables or disables robust automatic mode for distortion signal filtering in the RF input path.

return
automatic_mode: OFF | ON

get_weighting() → WeightingFilter

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:WEIGhting
value: enums.WeightingFilter = driver.configure.afRf.measurement.demodulation.
↪filterPy.get_weighting()
```

Selects the weighting filter in the RF input path.

return
weighting: OFF | AWEighting | CCITt | CMESsage OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter

set_deemphasis(*deemphasis: PreDeEmphasis*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DEEMphasis
driver.configure.afRf.measurement.demodulation.filterPy.set_
↪deemphasis(deemphasis = enums.PreDeEmphasis.OFF)
```

Configures the de-emphasis filter in the RF input path.

param deemphasis
OFF | T50 | T75 | T750 OFF Filter disabled T50, T75, T750 Time constant 50 us / 75 us / 750 us

set_enable(*disable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:ENABle
driver.configure.afRf.measurement.demodulation.filterPy.set_enable(disable =␣
↪False)
```

Selects whether the demodulation results are measured before or after the filters in the RF input path. For FM stereo, the demodulation results are always measured before the filters.

param disable
OFF | ON OFF Measure before filters ON Measure after filters

set_hpass(*highpass: HighpassFilterExtended*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:HPASs
driver.configure.afRf.measurement.demodulation.filterPy.set_hpass(highpass =␣
↪enums.HighpassFilterExtended.F300)
```

Configures the highpass filter in the RF input path.

param highpass
OFF | F6 | F50 | F300 OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz

set_lpass(*lowpass: LowpassFilterExtended*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:LPASs
driver.configure.afRf.measurement.demodulation.filterPy.set_lpass(lowpass =␣
↪enums.LowpassFilterExtended.F15K)
```

Configures the lowpass filter in the RF input path.

param lowpass
OFF | F255 | F3K | F3K4 | F4K | F15K OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

set_robust_auto(*automatic_mode: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:ROBustauto
driver.configure.afRf.measurement.demodulation.filterPy.set_robust_
↪auto(automatic_mode = False)
```

Enables or disables robust automatic mode for distortion signal filtering in the RF input path.

param automatic_mode
OFF | ON

set_weighting(*weighting: WeightingFilter*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:WEIGhting
driver.configure.afRf.measurement.demodulation.filterPy.set_weighting(weighting␣
↪= enums.WeightingFilter.AWEighting)
```

Selects the weighting filter in the RF input path.

param weighting
OFF | AWEighting | CCITt | CMESsage OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.filterPy.clone()
```

Subgroups

6.3.1.2.5.10 Bpass

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:BPASs:ENABle
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:BPASs:CFRequency
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:BPASs:BWIDth
```

class BpassCls

Bpass commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_bandwidth() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:BPASs:BWIDth
value: float = driver.configure.afRf.measurement.demodulation.filterPy.bpass.
↪get_bandwidth()
```

Configures the bandwidth of the variable bandpass filter in the RF input path.

return
bandwidth: Range: 20 Hz to 20 kHz, Unit: Hz

get_cfrequency() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FILTer:BPASs:CFRequency
value: float = driver.configure.afRf.measurement.demodulation.filterPy.bpass.
↪get_cfrequency()
```

Configures the center frequency of the variable bandpass filter in the RF input path.

return
center_freq: Range: 0 Hz to 21 kHz, Unit: Hz

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:BPASs:ENABle
value: bool = driver.configure.afRf.measurement.demodulation.filterPy.bpass.get_
↪enable()
```

Enables or disables the variable bandpass filter in the RF input path.

return
enable: OFF | ON

set_bandwidth(*bandwidth: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:BPASs:BWIDth
driver.configure.afRf.measurement.demodulation.filterPy.bpass.set_
↪bandwidth(bandwidth = 1.0)
```

Configures the bandwidth of the variable bandpass filter in the RF input path.

param bandwidth
Range: 20 Hz to 20 kHz, Unit: Hz

set_cfrequency(*center_freq: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FILTer:BPASs:CFRequency
driver.configure.afRf.measurement.demodulation.filterPy.bpass.set_
↪cfrequency(center_freq = 1.0)
```

Configures the center frequency of the variable bandpass filter in the RF input path.

param center_freq
Range: 0 Hz to 21 kHz, Unit: Hz

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:BPASs:ENABle
driver.configure.afRf.measurement.demodulation.filterPy.bpass.set_enable(enable␣
↪= False)
```

Enables or disables the variable bandpass filter in the RF input path.

param enable
OFF | ON

6.3.1.2.5.11 Dfrequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DFRequency
```

class DfrequencyCls

Dfrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DfrequencyStruct

Response structure. Fields:

- Distor_Freq_Left: float: Range: 0 Hz to 10.5 kHz, Unit: Hz
- Distor_Freq_Right: float: Range: 0 Hz to 10.5 kHz, Unit: Hz

get() → DfrequencyStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DFRequency
value: DfrequencyStruct = driver.configure.afRf.measurement.demodulation.
↪filterPy.dfrequency.get()
```

Configures the reference frequency for single-tone measurements via the RF input path. For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only <DistorFreqLeft> is relevant. <DistorFreqRight> has no effect.

return
structure: for return value, see the help for DfrequencyStruct structure arguments.

set(*distor_freq_left: float*, *distor_freq_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DFRequency
driver.configure.afRf.measurement.demodulation.filterPy.dfrequency.set(distor_
↪freq_left = 1.0, distor_freq_right = 1.0)
```

Configures the reference frequency for single-tone measurements via the RF input path. For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only <DistorFreqLeft> is relevant. <DistorFreqRight> has no effect.

param distor_freq_left
Range: 0 Hz to 10.5 kHz, Unit: Hz

param distor_freq_right
Range: 0 Hz to 10.5 kHz, Unit: Hz

6.3.1.2.5.12 Dwidth

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DWIDth
```

class DwidthCls

Dwidth commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class DwidthStruct

Response structure. Fields:

- Dwidth_Left: enums.PwrFilterType: WIDE | NARRow | UDEF Wide or narrow bandwidth
- Relative_Left: enums.Relative: RELative | CONStant Bandwidth proportional to reference frequency or constant
- Dwidth_Right: enums.PwrFilterType: WIDE | NARRow | UDEF
- Relative_Right: enums.Relative: RELative | CONStant

get() → DwidthStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DWIDth
value: DwidthStruct = driver.configure.afRf.measurement.demodulation.filterPy.
↪dwidth.get()
```

Configures the bandwidth of the distortion filter in the RF input path. For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only the <...Left> settings are relevant. The <... Right> settings have no effect.

return
structure: for return value, see the help for DwidthStruct structure arguments.

set(*dwidth_left: PwrFilterType*, *relative_left: Relative*, *dwidth_right: PwrFilterType*, *relative_right: Relative*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DWIDth
driver.configure.afRf.measurement.demodulation.filterPy.dwidth.set(dwidth_left␣
↪= enums.PwrFilterType.NARRow, relative_left = enums.Relative.CONStant, dwidth_
↪right = enums.PwrFilterType.NARRow, relative_right = enums.Relative.CONStant)
```

Configures the bandwidth of the distortion filter in the RF input path. For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only the <...Left> settings are relevant. The <... Right> settings have no effect.

param dwidth_left
WIDE | NARRow | UDEF Wide or narrow bandwidth

param relative_left
RELative | CONStant Bandwidth proportional to reference frequency or constant

param dwidth_right
WIDE | NARRow | UDEF

param relative_right
RELative | CONStant

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.filterPy.dwidth.clone()
```

Subgroups

6.3.1.2.5.13 Sfactor

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DWIDth:SFACtor
```

class SfactorCls

Sfactor commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SfactorStruct

Response structure. Fields:

- Factor_Left: float: Range: 0.001 to 0.005
- Factor_Right: float: Range: 0.001 to 0.005

get() → SfactorStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DWIDth:SFACtor
value: SfactorStruct = driver.configure.afRf.measurement.demodulation.filterPy.
↪dwidth.sfactor.get()
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:DEM:FILT:DWID UDEF For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only <FactorLeft> is relevant. <FactorRight> has no effect.

return
structure: for return value, see the help for SfactorStruct structure arguments.

set(*factor_left: float*, *factor_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:DWIDth:SFACtor
driver.configure.afRf.measurement.demodulation.filterPy.dwidth.sfactor.
↪set(factor_left = 1.0, factor_right = 1.0)
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:DEM:FILT:DWID UDEF For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only <FactorLeft> is relevant. <FactorRight> has no effect.

param factor_left
Range: 0.001 to 0.005

param factor_right
Range: 0.001 to 0.005

6.3.1.2.5.14 Notch<Notch>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.afRf.measurement.demodulation.filterPy.notch.repcap_notch_get()
driver.configure.afRf.measurement.demodulation.filterPy.notch.repcap_notch_set(repcap.
↪Notch.Nr1)
```

class NotchCls

Notch commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Notch, default value after init: Notch.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.filterPy.notch.clone()
```

Subgroups

6.3.1.2.5.15 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:NOTCh<Num>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*notch=Notch.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:NOTCh<Num>
↪:ENABle
value: bool = driver.configure.afRf.measurement.demodulation.filterPy.notch.
↪enable.get(notch = repcap.Notch.Default)
```

Enables the notch filters 1, 2 or 3 of the 'Demod' path.

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
notch_filter_enable: OFF | ON

set(*notch_filter_enable: bool*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:NOTCh<Num>
↪:ENABle
driver.configure.afRf.measurement.demodulation.filterPy.notch.enable.set(notch_
↪filter_enable = False, notch = repcap.Notch.Default)
```

Enables the notch filters 1, 2 or 3 of the 'Demod' path.

param notch_filter_enable
OFF | ON

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.5.16 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:NOTCh<Num>:FREQuency
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*notch=Notch.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:NOTCh<Num>
↪:FREQuency
value: float = driver.configure.afRf.measurement.demodulation.filterPy.notch.
↪frequency.get(notch = repcap.Notch.Default)
```

Sets the frequency for the notch filters 1, 2 or 3 of the 'Demod' path.

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
notch_filter_frequency: Range: 5 Hz to 21000 Hz, Unit: Hz

set(*notch_filter_frequency: float*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:NOTCh<Num>
↪:FREQuency
driver.configure.afRf.measurement.demodulation.filterPy.notch.frequency.
↪set(notch_filter_frequency = 1.0, notch = repcap.Notch.Default)
```

Sets the frequency for the notch filters 1, 2 or 3 of the 'Demod' path.

param notch_filter_frequency
Range: 5 Hz to 21000 Hz, Unit: Hz

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.5.17 FmStereo

class FmStereoCls

FmStereo 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.afRf.measurement.demodulation.fmStereo.clone()
```

Subgroups

6.3.1.2.5.18 Frequency

class FrequencyCls

Frequency 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.afRf.measurement.demodulation.fmStereo.frequency.clone()
```

Subgroups

6.3.1.2.5.19 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FMSTereo:FREQuency:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class MeasuredStruct

Structure for reading output parameters. Fields:

- Left_Meas_Val: float: Unit: Hz
- Right_Meas_Val: float: Unit: Hz

get_measured() → MeasuredStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FMSTereo:FREQuency:DELTa:MEASured
value: MeasuredStruct = driver.configure.afRf.measurement.demodulation.fmStereo.
↪frequency.delta.get_measured()
```

Configures the measured reference value.

return
structure: for return value, see the help for MeasuredStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.fmStereo.frequency.delta.clone()
```

Subgroups

6.3.1.2.5.20 User

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FMSTereo:FREQuency:DELTa:USER
```

class UserCls

User commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class UserStruct

Response structure. Fields:

- Left_User_Val: float: Unit: Hz
- Right_User_Val: float: Unit: Hz

get() → UserStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FMSTereo:FREQuency:DELTa:USER
value: UserStruct = driver.configure.afRf.measurement.demodulation.fmStereo.
↪frequency.delta.user.get()
```

Configures the user reference value.

return
structure: for return value, see the help for UserStruct structure arguments.

set(*left_user_val: float*, *right_user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FMSTereo:FREQuency:DELTa:USER
driver.configure.afRf.measurement.demodulation.fmStereo.frequency.delta.user.
↪set(left_user_val = 1.0, right_user_val = 1.0)
```

Configures the user reference value.

param left_user_val
Unit: Hz

param right_user_val
Unit: Hz

6.3.1.2.5.21 Frequency

class FrequencyCls

Frequency commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.frequency.clone()
```

Subgroups

6.3.1.2.5.22 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:DEModulation:FREQuency:DELTa:MEASured
value: float = driver.configure.afRf.measurement.demodulation.frequency.delta.
↪get_measured()
```

Configures the measured reference value.

return
meas_val: Unit: Hz

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.demodulation.
↪frequency.delta.get_mode()
```

Configures the reference mode for demodulation frequency.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:USER
value: float = driver.configure.afRf.measurement.demodulation.frequency.delta.
↪get_user()
```

Configures the user reference value.

return
user_val: Unit: Hz

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:MODE
driver.configure.afRf.measurement.demodulation.frequency.delta.set_mode(mode =␣
↪enums.DeltaMode.MEAS)
```

Configures the reference mode for demodulation frequency.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:USER
driver.configure.afRf.measurement.demodulation.frequency.delta.set_user(user_
↪val = 1.0)
```

Configures the user reference value.

param user_val
Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.frequency.delta.clone()
```

Subgroups

6.3.1.2.5.23 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.frequency.delta.update.set()
```

Triggers the update of the measurement reference value for demodulation frequency.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.frequency.delta.update.set_with_
↪opc()
```

Triggers the update of the measurement reference value for demodulation frequency.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.5.24 Gcoupling

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:GCOupling
```

class GcouplingCls

Gcoupling commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GcouplingStruct

Response structure. Fields:

- Coupling_Left: enums.GeneratorCoupling: OFF | GEN1 | GEN2 | GEN3 | GEN4 OFF No coupling GENn Coupled to audio generator n
- Coupling_Right: enums.GeneratorCoupling: OFF | GEN2 | GEN4

get() → GcouplingStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:GCOupling
value: GcouplingStruct = driver.configure.afRf.measurement.demodulation.
↪gcoupling.get()
```

Couples the audio output paths of the demodulator to an internal signal generator.

INTRO_CMD_HELP: For FM stereo, the settings configure the left and the right audio channel. Only the following combinations are allowed:

- OFF, OFF
- GEN1, GEN2
- GEN3, GEN4

For other modulation types, only <CouplingLeft> is relevant. <CouplingRight> has no effect.

return

structure: for return value, see the help for GcouplingStruct structure arguments.

set(*coupling_left: GeneratorCoupling*, *coupling_right: GeneratorCoupling*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:GCOupling
driver.configure.afRf.measurement.demodulation.gcoupling.set(coupling_left =␣
↪enums.GeneratorCoupling.GEN1, coupling_right = enums.GeneratorCoupling.GEN1)
```

Couples the audio output paths of the demodulator to an internal signal generator.

INTRO_CMD_HELP: For FM stereo, the settings configure the left and the right audio channel. Only the following combinations are allowed:

- OFF, OFF
- GEN1, GEN2
- GEN3, GEN4

For other modulation types, only <CouplingLeft> is relevant. <CouplingRight> has no effect.

param coupling_left

OFF | GEN1 | GEN2 | GEN3 | GEN4 OFF No coupling GENn Coupled to audio generator n

param coupling_right

OFF | GEN2 | GEN4

6.3.1.2.5.25 ModDepth

class ModDepthCls

ModDepth commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.modDepth.clone()
```

Subgroups

6.3.1.2.5.26 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:MEASured
value: float = driver.configure.afRf.measurement.demodulation.modDepth.delta.
↪get_measured()
```

Configures the measured reference value of the modulation depth for AM demodulation.

return
meas_val: Range: 0.01 % to 100.00 % , Unit: %

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.demodulation.
↪modDepth.delta.get_mode()
```

Sets the mode for the reference value of the modulation depth for AM demodulation.

return
mode: NONE | MEAS | USER NONE No reference value, delta measurement is disabled MEAS Measured reference value USER User-defined reference value

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:USER
value: float = driver.configure.afRf.measurement.demodulation.modDepth.delta.
↪get_user()
```

Configures the user-defined reference value of the modulation depth for AM demodulation.

return
user_val: Range: 0.01 % to 100.00 % , Unit: %

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:MODE
driver.configure.afRf.measurement.demodulation.modDepth.delta.set_mode(mode =␣
↪enums.DeltaMode.MEAS)
```

Sets the mode for the reference value of the modulation depth for AM demodulation.

param mode
NONE | MEAS | USER NONE No reference value, delta measurement is disabled MEAS Measured reference value USER User-defined reference value

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:USER
driver.configure.afRf.measurement.demodulation.modDepth.delta.set_user(user_val␣
↪= 1.0)
```

Configures the user-defined reference value of the modulation depth for AM demodulation.

param user_val
Range: 0.01 % to 100.00 % , Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.demodulation.modDepth.delta.clone()
```

Subgroups

6.3.1.2.5.27 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.modDepth.delta.update.set()
```

Triggers an update of the delta measurement modulation depth for AM demodulation.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:DELTa:UPDate
driver.configure.afRf.measurement.demodulation.modDepth.delta.update.set_with_
↪opc()
```

Triggers an update of the delta measurement modulation depth for AM demodulation.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.5.28 Tmode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DEModulation:TMODe
```

class TmodeCls

Tmode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TmodeStruct

Response structure. Fields:

- Tone_Mode_Left: enums.ToneMode: No parameter help available
- Tone_Mode_Right: enums.ToneMode: No parameter help available

get() → TmodeStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:TMODe
value: TmodeStruct = driver.configure.afRf.measurement.demodulation.tmode.get()
```

No command help available

return
structure: for return value, see the help for TmodeStruct structure arguments.

set(*tone_mode_left: ToneMode*, *tone_mode_right: ToneMode = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DEModulation:TMODe
driver.configure.afRf.measurement.demodulation.tmode.set(tone_mode_left = enums.
↪ToneMode.NOISe, tone_mode_right = enums.ToneMode.NOISe)
```

No command help available

param tone_mode_left
No help available

param tone_mode_right
No help available

6.3.1.2.6 Digital

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:SCOunt
CONFigure:AFRF:MEASurement<Instance>:DIGital:STANdard
CONFigure:AFRF:MEASurement<Instance>:DIGital:REPetition
CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONdition
CONFigure:AFRF:MEASurement<Instance>:DIGital:CREPetition
CONFigure:AFRF:MEASurement<Instance>:DIGital:MOEXception
CONFigure:AFRF:MEASurement<Instance>:DIGital:RCOupling
CONFigure:AFRF:MEASurement<Instance>:DIGital:TOUT
```

class DigitalCls

Digital commands group definition. 48 total commands, 8 Subgroups, 8 group commands

get_crepetition() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:CREPetition
value: bool = driver.configure.afRf.measurement.digital.get_crepetition()
```

Sets the repetition mode for BER measurement automatically to 'Continuous' if the local mode is used.

return
continuous_repetition: OFF | ON

get_mo_exception() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:MOEXception
value: bool = driver.configure.afRf.measurement.digital.get_mo_exception()
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

return
meas_on_exception: OFF | ON OFF Faulty results are rejected. ON Results are never rejected.

get_rcoupling() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:RCOupling
value: bool = driver.configure.afRf.measurement.digital.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupling: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:REPetition
value: enums.Repeat = driver.configure.afRf.measurement.digital.get_repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped

after the statistic count. CONTinuous Continuous measurement, running until explicitly terminated.

get_scondition() → StopCondition

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONdition
value: enums.StopCondition = driver.configure.afRf.measurement.digital.get_
↪scondition()
```

Selects whether the measurement is stopped after a failed limit check or continued.

return
stop_condition: NONE | SLFail NONE Continue measurement irrespective of the limit check. SLFail Stop measurement on limit failure.

get_scount() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:SCOunt
value: int = driver.configure.afRf.measurement.digital.get_scount()
```

Sets the number of measurement intervals per measurement cycle.

return
statistic_count: No help available

get_standard() → StandardDigital

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:STANdard
value: enums.StandardDigital = driver.configure.afRf.measurement.digital.get_
↪standard()
```

Selects the digital standard of the measured signal.

return
standard: DMR | TETRa | PTFive DMR Digital mobile radio (DMR) TETRa Terrestrial Trunked Radio (TETRA) PTFive Project 25 Phase 1, P25, APCO-P25

get_timeout() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TOUT
value: float = driver.configure.afRf.measurement.digital.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_crepetition(*continuous_repetition: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:CREPetition
driver.configure.afRf.measurement.digital.set_crepetition(continuous_repetition␣
↪= False)
```

Sets the repetition mode for BER measurement automatically to 'Continuous' if the local mode is used.

param continuous_repetition
OFF | ON

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:MOEXception
driver.configure.afRf.measurement.digital.set_mo_exception(meas_on_exception =␣
↪False)
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

param meas_on_exception
OFF | ON OFF Faulty results are rejected. ON Results are never rejected.

set_rcoupling(*repetition_coupling: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:RCOupling
driver.configure.afRf.measurement.digital.set_rcoupling(repetition_coupling =␣
↪False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupling
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:REPetition
driver.configure.afRf.measurement.digital.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after the statistic count. CONTinuous Continuous measurement, running until explicitly terminated.

set_scondition(*stop_condition: StopCondition*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONdition
driver.configure.afRf.measurement.digital.set_scondition(stop_condition = enums.
↪StopCondition.NONE)
```

Selects whether the measurement is stopped after a failed limit check or continued.

param stop_condition
NONE | SLFail NONE Continue measurement irrespective of the limit check. SLFail Stop measurement on limit failure.

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:SCOunt
driver.configure.afRf.measurement.digital.set_scount(statistic_count = 1)
```

Sets the number of measurement intervals per measurement cycle.

param statistic_count
No help available

set_standard(*standard: StandardDigital*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:STANdard
driver.configure.afRf.measurement.digital.set_standard(standard = enums.
↪StandardDigital.DMR)
```

Selects the digital standard of the measured signal.

param standard
DMR | TETRa | PTFive DMR Digital mobile radio (DMR) TETRa Terrestrial Trunked Radio (TETRA) PTFive Project 25 Phase 1, P25, APCO-P25

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TOUT
driver.configure.afRf.measurement.digital.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.digital.clone()
```

Subgroups

6.3.1.2.6.1 Dmr

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:LDIRection
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:PTYPe
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:BERPeriod
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:CMODe
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:FILTer
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:ROFactor
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:SRATe
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:SDEViation
CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:MODE
```

class DmrCls

Dmr commands group definition. 9 total commands, 0 Subgroups, 9 group commands

get_ber_period() → BerPeriod

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:BERPeriod
value: enums.BerPeriod = driver.configure.afRf.measurement.digital.dmr.get_ber_
↪period()
```

Sets the number of frames for the BER measurement.

return
ber_period: F36 | F48

get_cmode() → ChannelModeDmr

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:CMODe
value: enums.ChannelModeDmr = driver.configure.afRf.measurement.digital.dmr.get_
↪cmode()
```

Specifies if 'Voice' or 'Data' is transmitted over the radio channel. Currently, only 'Voice' is supported.

return
channel_mode: VOICe | DATA

get_filter_py() → FilterDigital

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:FILTer
value: enums.FilterDigital = driver.configure.afRf.measurement.digital.dmr.get_
↪filter_py()
```

Selects the filter type for pulse shaping of DMR.

return
filter_py: GAUSs | RRC | COSine | SINC

get_ldirection() → LinkDirectionDmr

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:LDIRection
value: enums.LinkDirectionDmr = driver.configure.afRf.measurement.digital.dmr.
↪get_ldirection()
```

Specifies the direction of voice/data transmission. The details of the used frames depend on this selection.

return
link_dirction: MSSourced

get_mode() → DemodulationType

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:MODE
value: enums.DemodulationType = driver.configure.afRf.measurement.digital.dmr.
↪get_mode()
```

Queries the modulation type used for DMR.

return
mode: FSK4

get_ptype() → DmrPatternB

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:PTYPe
value: enums.DmrPatternB = driver.configure.afRf.measurement.digital.dmr.get_
↪ptype()
```

Selects the expected payload type that can be a bit pattern or a signal.

return
payload_type: P1031 | SYNC | SILence

get_ro_factor() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:ROFactor
value: float = driver.configure.afRf.measurement.digital.dmr.get_ro_factor()
```

Sets the roll-off factor of the filter used for pulse shaping of DMR.

return
rolloff_factor: Range: 0 to 1

get_standard_dev() → List[float]

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:SDEViation
value: List[float] = driver.configure.afRf.measurement.digital.dmr.get_standard_
↪dev()
```

Queries the frequency deviations of the 4FSK modulation for DMR.

return
sdeviation: List of four frequency deviations, for the symbols 01, 00, 10, 11. Range: -2000 Hz to 2000 Hz, Unit: Hz

get_symbol_rate() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:SRATe
value: float = driver.configure.afRf.measurement.digital.dmr.get_symbol_rate()
```

Queries the symbol rate for DMR.

return
srate: Range: 1 symbol/s to 100E+6 symbol/s, Unit: symbol/s

set_ber_period(*ber_period: BerPeriod*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:BERPeriod
driver.configure.afRf.measurement.digital.dmr.set_ber_period(ber_period = enums.
↪BerPeriod.F36)
```

Sets the number of frames for the BER measurement.

param ber_period
F36 | F48

set_ldirection(*link_dirction: LinkDirectionDmr*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:LDIRection
driver.configure.afRf.measurement.digital.dmr.set_ldirection(link_dirction =␣
↪enums.LinkDirectionDmr.MSSourced)
```

Specifies the direction of voice/data transmission. The details of the used frames depend on this selection.

param link_dirction
MSSourced

set_ptype(*payload_type: DmrPatternB*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:DMR:PTYPe
driver.configure.afRf.measurement.digital.dmr.set_ptype(payload_type = enums.
↪DmrPatternB.P1031)
```

Selects the expected payload type that can be a bit pattern or a signal.

param payload_type
P1031 | SYNC | SILence

6.3.1.2.6.2 Limit

class LimitCls

Limit 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.configure.afRf.measurement.digital.limit.clone()
```

Subgroups

6.3.1.2.6.3 Dmr

class DmrCls

Dmr 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.afRf.measurement.digital.limit.dmr.clone()
```

Subgroups

6.3.1.2.6.4 BitErrorRate

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:DMR:BERate
```

class BitErrorRateCls

BitErrorRate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class BitErrorRateStruct

Response structure. Fields:

- Enable_Limit: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper limit Range: 0.1 % to 100 %, Unit: %

get() → BitErrorRateStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:DMR:BERate
value: BitErrorRateStruct = driver.configure.afRf.measurement.digital.limit.dmr.
↪bitErrorRate.get()
```

Configures the upper limit of the bit error rate for standard DMR.

return
structure: for return value, see the help for BitErrorRateStruct structure arguments.

set(*enable_limit: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:DMR:BERate
driver.configure.afRf.measurement.digital.limit.dmr.bitErrorRate.set(enable_
↪limit = False, upper = 1.0)
```

Configures the upper limit of the bit error rate for standard DMR.

param enable_limit
OFF | ON Enables or disables the limit check

param upper
Upper limit Range: 0.1 % to 100 %, Unit: %

6.3.1.2.6.5 PtFive

class PtFiveCls

PtFive 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.afRf.measurement.digital.limit.ptFive.clone()
```

Subgroups

6.3.1.2.6.6 BitErrorRate

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:PTFive:BERate
```

class BitErrorRateCls

BitErrorRate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class BitErrorRateStruct

Response structure. Fields:

- Enable_Limit: bool: No parameter help available
- Upper: float: Upper limit Range: 0.1 % to 100 %, Unit: %

get() → BitErrorRateStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:PTFive:BERate
value: BitErrorRateStruct = driver.configure.afRf.measurement.digital.limit.
↪ptFive.bitErrorRate.get()
```

Configures the upper limit of the bit error rate for standard P25.

return
structure: for return value, see the help for BitErrorRateStruct structure arguments.

set(*enable_limit: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:PTFive:BERate
driver.configure.afRf.measurement.digital.limit.ptFive.bitErrorRate.set(enable_
↪limit = False, upper = 1.0)
```

Configures the upper limit of the bit error rate for standard P25.

param enable_limit
OFF | ON Enables or disables the limit check

param upper
Upper limit Range: 0.1 % to 100 %, Unit: %

6.3.1.2.6.7 Tetra

class TetraCls

Tetra commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.digital.limit.tetra.clone()
```

Subgroups

6.3.1.2.6.8 BitErrorRate

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TETRa:BERate
```

class BitErrorRateCls

BitErrorRate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class BitErrorRateStruct

Response structure. Fields:

- Enable_Limit: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper limit Range: 0.1 % to 100 %, Unit: %

get() → BitErrorRateStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TETRa:BERate
value: BitErrorRateStruct = driver.configure.afRf.measurement.digital.limit.
↪tetra.bitErrorRate.get()
```

Configures the upper limit of the bit error rate for standard TETRA.

return
structure: for return value, see the help for BitErrorRateStruct structure arguments.

set(*enable_limit: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TETRa:BERate
driver.configure.afRf.measurement.digital.limit.tetra.bitErrorRate.set(enable_
↪limit = False, upper = 1.0)
```

Configures the upper limit of the bit error rate for standard TETRA.

param enable_limit
OFF | ON Enables or disables the limit check

param upper
Upper limit Range: 0.1 % to 100 %, Unit: %

6.3.1.2.6.9 FreqError

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TETRa:FERRor
```

class FreqErrorCls

FreqError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FreqErrorStruct

Response structure. Fields:

- Enable_Limit: bool: No parameter help available
- Upper: float: No parameter help available
- Lower: float: No parameter help available

get() → FreqErrorStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TETRa:FERRor
value: FreqErrorStruct = driver.configure.afRf.measurement.digital.limit.tetra.
↪freqError.get()
```

No command help available

return
structure: for return value, see the help for FreqErrorStruct structure arguments.

set(*enable_limit: bool*, *upper: float*, *lower: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TETRa:FERRor
driver.configure.afRf.measurement.digital.limit.tetra.freqError.set(enable_
↪limit = False, upper = 1.0, lower = 1.0)
```

No command help available

param enable_limit
No help available

param upper
No help available

param lower
No help available

6.3.1.2.6.10 Ttl

class TtlCls

Ttl 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.afRf.measurement.digital.limit.ttl.clone()
```

Subgroups

6.3.1.2.6.11 BitErrorRate

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TTL:BERate
```

class BitErrorRateCls

BitErrorRate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class BitErrorRateStruct

Response structure. Fields:

- Enable_Limit: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper limit Range: 0.1 % to 100 %, Unit: %

get() → BitErrorRateStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TTL:BERate
value: BitErrorRateStruct = driver.configure.afRf.measurement.digital.limit.ttl.
↪bitErrorRate.get()
```

Configures the upper limit of the bit error rate for the TTL path.

return
structure: for return value, see the help for BitErrorRateStruct structure arguments.

`set`(*enable_limit: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:LIMit:TTL:BERate
driver.configure.afRf.measurement.digital.limit.ttl.bitErrorRate.set(enable_
↪limit = False, upper = 1.0)
```

Configures the upper limit of the bit error rate for the TTL path.

param enable_limit
OFF | ON Enables or disables the limit check

param upper
Upper limit Range: 0.1 % to 100 %, Unit: %

6.3.1.2.6.12 PtFive

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:PTFive:PTYPe
```

`class PtFiveCls`

PtFive commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`get_ptype`() → PayloadType

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:PTFive:PTYPe
value: enums.PayloadType = driver.configure.afRf.measurement.digital.ptFive.get_
↪ptype()
```

Defines the payload type for P25 digital standard.

return
payload_type: P1011 | SILence 1011 Audio tone with a frequency of 1011 Hz. SILence The payload contains silence.

6.3.1.2.6.13 Result

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:RESult:BER
```

`class ResultCls`

Result commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`get_ber`() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:RESult:BER
value: bool = driver.configure.afRf.measurement.digital.result.get_ber()
```

Enables or disables the indication of the BER measurement results.

return
ber_enable: OFF | ON

set_ber(*ber_enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:RESult:BER
driver.configure.afRf.measurement.digital.result.set_ber(ber_enable = False)
```

Enables or disables the indication of the BER measurement results.

param ber_enable
OFF | ON

6.3.1.2.6.14 Rf

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:RF:ENABle
```

class RfCls

Rf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:RF:ENABle
value: bool = driver.configure.afRf.measurement.digital.rf.get_enable()
```

Enables or disables the RF input path.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:RF:ENABle
driver.configure.afRf.measurement.digital.rf.set_enable(enable = False)
```

Enables or disables the RF input path.

param enable
OFF | ON

6.3.1.2.6.15 Sync

class SyncCls

Sync 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.afRf.measurement.digital.sync.clone()
```

Subgroups

6.3.1.2.6.16 Timeout

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:SYNC:TOUT
```

class TimeoutCls

Timeout commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TimeoutStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Timeout: float: Unit: s

get() → TimeoutStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:SYNC:TOUT
value: TimeoutStruct = driver.configure.afRf.measurement.digital.sync.timeout.
↪get()
```

Synchronizes the timeout after a defined period.

return
structure: for return value, see the help for TimeoutStruct structure arguments.

set(*enable: bool*, *timeout: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:SYNC:TOUT
driver.configure.afRf.measurement.digital.sync.timeout.set(enable = False,␣
↪timeout = 1.0)
```

Synchronizes the timeout after a defined period.

param enable
OFF | ON

param timeout
Unit: s

6.3.1.2.6.17 Tetra

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:SRATe
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:SDEViation
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:MODE
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:LDIRection
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:PATTern
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:BERPeriod
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:PTYPe
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:CTYPe
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:TMODe
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:CMODe
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:RPRBs
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:FILTer
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:ROFactor
```

class TetraCls

Tetra commands group definition. 16 total commands, 1 Subgroups, 13 group commands

get_ber_period() → BerPeriod

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:BERPeriod
value: enums.BerPeriod = driver.configure.afRf.measurement.digital.tetra.get_
↪ber_period()
```

Sets the period, i.e. the number of frames for the bit error rate. Select 36 or 48 frames.

return
ber_period: F36 | F48 36 Frames The bit error rate is calculated from 36 frames of the bit stream. 48 Frames The bit error rate is calculated from 48 frames of the bit stream.

get_cmode() → ChannelModeTetra

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:CMODe
value: enums.ChannelModeTetra = driver.configure.afRf.measurement.digital.tetra.
↪get_cmode()
```

The TCH 7.2 traffic channel mode is preset.

return
channel_mode: TCH72

get_ctype() → ChannelTypeTetra

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:CTYPe
value: enums.ChannelTypeTetra = driver.configure.afRf.measurement.digital.tetra.
↪get_ctype()
```

Sets the channel type. It is fixed to 0.

return
channel_type: CT0 | CT1 | CT2 | CT3 | CT4 | CT21 | CT22 | CT24

get_filter_py() → FilterDigital

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:FILTer
value: enums.FilterDigital = driver.configure.afRf.measurement.digital.tetra.
↪get_filter_py()
```

Selects the filter type for pulse shaping of TETRA.

return
filter_py: GAUSs | RRC | COSine | SINC

get_ldirection() → LinkDirectionTetra

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:LDIRection
value: enums.LinkDirectionTetra = driver.configure.afRf.measurement.digital.
↪tetra.get_ldirection()
```

Sets either the Downlink/forward or the uplink/backward direction of the test. The downlink direction is from BS to MS, the uplink direction vice versa.

return
link_dirction: DLNK | ULNK Downlink/Forward Direction from BS to MS. Uplink/Backward Direction from MS to BS.

get_mode() → ModeTetra

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:MODE
value: enums.ModeTetra = driver.configure.afRf.measurement.digital.tetra.get_
↪mode()
```

Queries the modulation type used for TETRA.

return
mode: DQPSK

get_pattern() → PatternTetra

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:PATTern
value: enums.PatternTetra = driver.configure.afRf.measurement.digital.tetra.get_
↪pattern()
```

Selects the pattern type.

return
pattern: S1 | S2 | S3

get_ptype() → PayloadTypeTetra

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:PTYPe
value: enums.PayloadTypeTetra = driver.configure.afRf.measurement.digital.tetra.
↪get_ptype()
```

Defines the payload type for TETRA digital standard.

return
payload_type: ALLZero | ALLO | ALTE | PRBS9 | USER AllZero The payload contains a binary sequence of all 0. AllOnes The payload contains a binary sequence of all 1. ALTErnating The payload contains a binary sequence with alternating 0 and 1.

PRBS9 The payload contains a pseudo-random binary sequence with 511 bits (29-1) .
USER The payload contains a user-defined binary sequence.

get_ro_factor() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:ROFactor
value: float = driver.configure.afRf.measurement.digital.tetra.get_ro_factor()
```

Sets the roll-off factor of the filter used for pulse shaping of TETRA.

return
rolloff_factor: Range: 0 to 1

get_rprbs() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:RPRBs
value: bool = driver.configure.afRf.measurement.digital.tetra.get_rprbs()
```

Resets the PRBS bit pattern at frame 0.

return
reset_prbs_at_fm_zero: OFF | ON

get_standard_dev() → List[str]

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:SDEViation
value: List[str] = driver.configure.afRf.measurement.digital.tetra.get_standard_
↪dev()
```

Queries the phase changes of the DQPSK modulation for TETRA.

return
sdeviation: List of four phase changes, for the symbols 01, 00, 10, 11.

get_symbol_rate() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:SRATe
value: float = driver.configure.afRf.measurement.digital.tetra.get_symbol_rate()
```

Queries the symbol rate for TETRA.

return
srate: Unit: symbol/s

get_tmode() → TestModeTetra

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:TMODe
value: enums.TestModeTetra = driver.configure.afRf.measurement.digital.tetra.
↪get_tmode()
```

Sets the test mode. The T1 test mode is fixed.

return
test_mode: VSE | T1 | SIDecoding

set_ber_period(*ber_period: BerPeriod*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:BERPeriod
driver.configure.afRf.measurement.digital.tetra.set_ber_period(ber_period =␣
↪enums.BerPeriod.F36)
```

Sets the period, i.e. the number of frames for the bit error rate. Select 36 or 48 frames.

param ber_period
F36 | F48 36 Frames The bit error rate is calculated from 36 frames of the bit stream. 48 Frames The bit error rate is calculated from 48 frames of the bit stream.

set_cmode(*channel_mode: ChannelModeTetra*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:CMODe
driver.configure.afRf.measurement.digital.tetra.set_cmode(channel_mode = enums.
↪ChannelModeTetra.TCH72)
```

The TCH 7.2 traffic channel mode is preset.

param channel_mode
TCH72

set_ctype(*channel_type: ChannelTypeTetra*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:CTYPe
driver.configure.afRf.measurement.digital.tetra.set_ctype(channel_type = enums.
↪ChannelTypeTetra.CT0)
```

Sets the channel type. It is fixed to 0.

param channel_type
CT0 | CT1 | CT2 | CT3 | CT4 | CT21 | CT22 | CT24

set_ldirection(*link_dirction: LinkDirectionTetra*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:LDIRection
driver.configure.afRf.measurement.digital.tetra.set_ldirection(link_dirction =␣
↪enums.LinkDirectionTetra.DLNK)
```

Sets either the Downlink/forward or the uplink/backward direction of the test. The downlink direction is from BS to MS, the uplink direction vice versa.

param link_dirction
DLNK | ULNK Downlink/Forward Direction from BS to MS. Uplink/Backward Direction from MS to BS.

set_rprbs(*reset_prbs_at_fm_zero: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:RPRBs
driver.configure.afRf.measurement.digital.tetra.set_rprbs(reset_prbs_at_fm_zero␣
↪= False)
```

Resets the PRBS bit pattern at frame 0.

param reset_prbs_at_fm_zero
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.digital.tetra.clone()
```

Subgroups

6.3.1.2.6.18 Uplink

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:BCC
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:MCC
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:MNC
```

class UplinkCls

Uplink commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_bcc() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:BCC
value: int = driver.configure.afRf.measurement.digital.tetra.uplink.get_bcc()
```

Defines for the TETRA standard the color code to be signaled from the base station to the DUT.

return
base_color_code: No help available

get_mcc() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:MCC
value: int = driver.configure.afRf.measurement.digital.tetra.uplink.get_mcc()
```

Sets the mobile country code for TETRA.

return
mcc: No help available

get_mnc() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:MNC
value: int = driver.configure.afRf.measurement.digital.tetra.uplink.get_mnc()
```

Sets the mobile network code for TETRA.

return
mnc: No help available

set_bcc(*base_color_code: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:BCC
driver.configure.afRf.measurement.digital.tetra.uplink.set_bcc(base_color_code␣
↪= 1)
```

Defines for the TETRA standard the color code to be signaled from the base station to the DUT.

param base_color_code
No help available

set_mcc(*mcc: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:MCC
driver.configure.afRf.measurement.digital.tetra.uplink.set_mcc(mcc = 1)
```

Sets the mobile country code for TETRA.

param mcc
No help available

set_mnc(*mnc: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:MNC
driver.configure.afRf.measurement.digital.tetra.uplink.set_mnc(mnc = 1)
```

Sets the mobile network code for TETRA.

param mnc
No help available

6.3.1.2.6.19 Ttl

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:PATTern
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:ENABle
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:INTerface
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:CRATe
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:BPFactor
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:BPBits
```

class TtlCls

Ttl commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_bp_bits() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:BPBits
value: int = driver.configure.afRf.measurement.digital.ttl.get_bp_bits()
```

Provides the number of bits for the 'BER Period'. The number of bits is a multiple of the clock rate. Set the factor in the 'BER Period' input field or in the following remote command: method RsCma.Configure.AfRf.Measurement.Digital.Ttl. bpFactor.

return
ber_period: Range: 1200 bits to 24E+3 bits, Unit: bits

get_bp_factor() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:BPFactor
value: int = driver.configure.afRf.measurement.digital.ttl.get_bp_factor()
```

Sets the factor for the 'BER Period'. The factor sets the number of bits which are a multiple of the clock rate.

return
bperiod_factor: Range: 1 to 20

get_crate() → ClockRate

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:CRATe
value: enums.ClockRate = driver.configure.afRf.measurement.digital.ttl.get_
↪crate()
```

Sets the clock rate.

return
clock_rate: BPS1200 | BPS2400 | BPS4800 | BPS9600 | BPS14400 | BPS19200 | BPS28800 | BPS38400 | BPS57600 | BPS115200

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:ENABle
value: bool = driver.configure.afRf.measurement.digital.ttl.get_enable()
```

Enables or disables analysis of data from the TTL connector.

return
enable: OFF | ON

get_interface() → TtlInterface

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:INTerface
value: enums.TtlInterface = driver.configure.afRf.measurement.digital.ttl.get_
↪interface()
```

Sets '1-Wire' or '2-Wire' for the 'Interface' options field in the TTL path.

return
interface: WIRE1 | WIRE2

get_pattern() → UserDefPattern

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:PATTern
value: enums.UserDefPattern = driver.configure.afRf.measurement.digital.ttl.get_
↪pattern()
```

Selects the bit pattern to be transmitted as payload.

return
pattern: PRBS6 | PRBS9

set_bp_factor(*bperiod_factor: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:BPFactor
driver.configure.afRf.measurement.digital.ttl.set_bp_factor(bperiod_factor = 1)
```

Sets the factor for the 'BER Period'. The factor sets the number of bits which are a multiple of the clock rate.

param bperiod_factor
Range: 1 to 20

set_crate(*clock_rate: ClockRate*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:CRATe
driver.configure.afRf.measurement.digital.ttl.set_crate(clock_rate = enums.
↪ClockRate.BPS115200)
```

Sets the clock rate.

param clock_rate
BPS1200 | BPS2400 | BPS4800 | BPS9600 | BPS14400 | BPS19200 | BPS28800 | BPS38400 | BPS57600 | BPS115200

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:ENABle
driver.configure.afRf.measurement.digital.ttl.set_enable(enable = False)
```

Enables or disables analysis of data from the TTL connector.

param enable
OFF | ON

set_interface(*interface: TtlInterface*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:INTerface
driver.configure.afRf.measurement.digital.ttl.set_interface(interface = enums.
↪TtlInterface.WIRE1)
```

Sets '1-Wire' or '2-Wire' for the 'Interface' options field in the TTL path.

param interface
WIRE1 | WIRE2

set_pattern(*pattern: UserDefPattern*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:PATTern
driver.configure.afRf.measurement.digital.ttl.set_pattern(pattern = enums.
↪UserDefPattern.PRBS6)
```

Selects the bit pattern to be transmitted as payload.

param pattern
PRBS6 | PRBS9

6.3.1.2.7 FilterPy

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:FILTer:DISable
```

class FilterPyCls

FilterPy commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_disable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FILTer:DISable
value: bool = driver.configure.afRf.measurement.filterPy.get_disable()
```

Disables all filters for all paths including notch filters.

return
disable: OFF | ON

set_disable(*disable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FILTer:DISable
driver.configure.afRf.measurement.filterPy.set_disable(disable = False)
```

Disables all filters for all paths including notch filters.

param disable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.filterPy.clone()
```

Subgroups

6.3.1.2.7.1 Notch

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:FILTer:NOTCh:PATH
```

class NotchCls

Notch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_path() → NotchPath

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FILTer:NOTCh:PATH
value: enums.NotchPath = driver.configure.afRf.measurement.filterPy.notch.get_
↪path()
```

Selects the active path for the notch filters in the 'AF IN', 'SPDIF IN' and 'VoIP' subtabs.

return
path: AF | SPDIF | VoIP

set_path(*path: NotchPath*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FILTer:NOTCh:PATH
driver.configure.afRf.measurement.filterPy.notch.set_path(path = enums.
↪NotchPath.AF)
```

Selects the active path for the notch filters in the 'AF IN', 'SPDIF IN' and 'VoIP' subtabs.

param path
AF | SPDIF | VoIP

6.3.1.2.8 Frequency

class FrequencyCls

Frequency commands group definition. 14 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.frequency.clone()
```

Subgroups

6.3.1.2.8.1 Counter

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:DETection
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AFRequency
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:GCOupling
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:REPetition
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:TOUT
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:SPOWer
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AUTomatic
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:MODE
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FEPower
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:BURSt
```

class CounterCls

Counter commands group definition. 14 total commands, 2 Subgroups, 10 group commands

get_afrequency() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AFRequency
value: float = driver.configure.afRf.measurement.frequency.counter.get_
↪afrequency()
```

Specifies the single-tone audio frequency. Only relevant for SSB.

return
audio: Range: 1 Hz to 10500 Hz, Unit: Hz

get_automatic() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AUTomatic
value: bool = driver.configure.afRf.measurement.frequency.counter.get_
↪automatic()
```

Selects whether search results found in 'SingleShot' mode are applied automatically or not.

return
auto: OFF | ON

get_burst() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:BURSt
value: bool = driver.configure.afRf.measurement.frequency.counter.get_burst()
```

Enable burst signal mode for improved detection of pulsed signals or burst signal with the 'Find RF' measurement procedure.

return
state: OFF | ON

get_detection() → Repeat

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:DETection
value: enums.Repeat = driver.configure.afRf.measurement.frequency.counter.get_
↪detection()
```

Selects whether the search procedure stops after finding an RF signal, or continues.

return
detection: SINGleshot | CONTinuous SINGleshot If the search procedure finds a signal during a search cycle, it stops after the cycle. CONTinuous The procedure continues searching until you abort the search.

get_fe_power() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FEPower
value: bool = driver.configure.afRf.measurement.frequency.counter.get_fe_power()
```

Fixes the 'Expected Power' for the search procedure.

return
state: OFF | ON

get_gcoupling() → GeneratorCoupling

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:GCOupling
value: enums.GeneratorCoupling = driver.configure.afRf.measurement.frequency.
↪counter.get_gcoupling()
```

Couples the single-tone audio frequency to an internal signal generator. Only relevant for SSB.

return
coupling: OFF | GEN1 | GEN2 | GEN3 | GEN4 OFF No coupling GENn Coupled to audio generator n

get_mode() → AnalogDigital

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:MODE
value: enums.AnalogDigital = driver.configure.afRf.measurement.frequency.
↪counter.get_mode()
```

Selects, whether the search procedure is used in 'Analog' or 'Digital' scenarios.

return
mode: ANALog | DIGital

get_repetition() → Repeat

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:REPetition
value: enums.Repeat = driver.configure.afRf.measurement.frequency.counter.get_
↪repetition()
```

No command help available

return
repetition_mode: No help available

get_spower() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:SPOWer
value: float = driver.configure.afRf.measurement.frequency.counter.get_spower()
```

No command help available

return
search_power: No help available

get_timeout() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:TOUT
value: int = driver.configure.afRf.measurement.frequency.counter.get_timeout()
```

Specifies a timeout for the search procedure.

return
timeout: Range: 0 s to 36E+3 s, Unit: s

set_afrequency(*audio: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AFRequency
driver.configure.afRf.measurement.frequency.counter.set_afrequency(audio = 1.0)
```

Specifies the single-tone audio frequency. Only relevant for SSB.

param audio
Range: 1 Hz to 10500 Hz, Unit: Hz

set_automatic(*auto: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AUTomatic
driver.configure.afRf.measurement.frequency.counter.set_automatic(auto = False)
```

Selects whether search results found in 'SingleShot' mode are applied automatically or not.

param auto
OFF | ON

set_burst(*state: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:BURSt
driver.configure.afRf.measurement.frequency.counter.set_burst(state = False)
```

Enable burst signal mode for improved detection of pulsed signals or burst signal with the 'Find RF' measurement procedure.

param state
OFF | ON

set_detection(*detection: Repeat*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:DETection
driver.configure.afRf.measurement.frequency.counter.set_detection(detection =␣
↪enums.Repeat.CONTinuous)
```

Selects whether the search procedure stops after finding an RF signal, or continues.

param detection
SINGleshot | CONTinuous SINGleshot If the search procedure finds a signal during a search cycle, it stops after the cycle. CONTinuous The procedure continues searching until you abort the search.

set_fe_power(*state: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FEPower
driver.configure.afRf.measurement.frequency.counter.set_fe_power(state = False)
```

Fixes the 'Expected Power' for the search procedure.

param state
OFF | ON

set_gcoupling(*coupling: GeneratorCoupling*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:GCOupling
driver.configure.afRf.measurement.frequency.counter.set_gcoupling(coupling =␣
↪enums.GeneratorCoupling.GEN1)
```

Couples the single-tone audio frequency to an internal signal generator. Only relevant for SSB.

param coupling
OFF | GEN1 | GEN2 | GEN3 | GEN4 OFF No coupling GENn Coupled to audio generator n

set_mode(*mode: AnalogDigital*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:MODE
driver.configure.afRf.measurement.frequency.counter.set_mode(mode = enums.
↪AnalogDigital.ANALog)
```

Selects, whether the search procedure is used in 'Analog' or 'Digital' scenarios.

param mode
ANALog | DIGital

set_repetition(*repetition_mode: Repeat*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:REPetition
driver.configure.afRf.measurement.frequency.counter.set_repetition(repetition_
↪mode = enums.Repeat.CONTinuous)
```

No command help available

param repetition_mode
No help available

set_spower(*search_power: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:SPOWer
driver.configure.afRf.measurement.frequency.counter.set_spower(search_power = 1.
↪0)
```

No command help available

param search_power
No help available

set_timeout(*timeout: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:TOUT
driver.configure.afRf.measurement.frequency.counter.set_timeout(timeout = 1)
```

Specifies a timeout for the search procedure.

param timeout
Range: 0 s to 36E+3 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.frequency.counter.clone()
```

Subgroups

6.3.1.2.8.2 Frange

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:STOP
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:STARt
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:ENABle
```

class FrangeCls

Frange commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:ENABle
value: bool = driver.configure.afRf.measurement.frequency.counter.frange.get_
↪enable()
```

Selects the frequency range to be searched.

return
to_limit: OFF | ON OFF Entire supported frequency range ON Frequency range defined by the commands method RsCma.Configure.AfRf.Measurement.Frequency.Counter.Frange.start and method RsCma.Configure.AfRf.Measurement.Frequency.Counter.Frange.stop

get_start() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:STARt
value: float = driver.configure.afRf.measurement.frequency.counter.frange.get_
↪start()
```

Defines the minimum frequency for the search procedure.

return
lower: Range: 0 Hz to 3 GHz, Unit: Hz

get_stop() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:STOP
value: float = driver.configure.afRf.measurement.frequency.counter.frange.get_
↪stop()
```

Defines the maximum frequency for the search procedure.

return
upper: Range: 0 Hz to 3 GHz, Unit: Hz

set_enable(*to_limit: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:ENABle
driver.configure.afRf.measurement.frequency.counter.frange.set_enable(to_limit␣
↪= False)
```

Selects the frequency range to be searched.

param to_limit
OFF | ON OFF Entire supported frequency range ON Frequency range defined by the commands method RsCma.Configure.AfRf.Measurement.Frequency.Counter.Frange.start and method RsCma.Configure.AfRf.Measurement.Frequency.Counter.Frange.stop

set_start(*lower: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:STARt
driver.configure.afRf.measurement.frequency.counter.frange.set_start(lower = 1.
↪0)
```

Defines the minimum frequency for the search procedure.

param lower
Range: 0 Hz to 3 GHz, Unit: Hz

set_stop(*upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FRANge:STOP
driver.configure.afRf.measurement.frequency.counter.frange.set_stop(upper = 1.0)
```

Defines the maximum frequency for the search procedure.

param upper
Range: 0 Hz to 3 GHz, Unit: Hz

6.3.1.2.8.3 Use

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:USE
```

class UseCls

Use commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:USE
driver.configure.afRf.measurement.frequency.counter.use.set()


    INTRO_CMD_HELP: Applies the search results to the RF settings:

    - The center frequency of the RF analyzer is set to the counted frequency.
    - The expected power is set to the measured power plus 10 dB.
```

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:USE
driver.configure.afRf.measurement.frequency.counter.use.set_with_opc()


    INTRO_CMD_HELP: Applies the search results to the RF settings:

    - The center frequency of the RF analyzer is set to the counted frequency.
    - The expected power is set to the measured power plus 10 dB.
```

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.9 MultiEval

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:CREPetition
CONFigure:AFRF:MEASurement<Instance>:MEValuation:REPetition
CONFigure:AFRF:MEASurement<Instance>:MEValuation:SCONdition
CONFigure:AFRF:MEASurement<Instance>:MEValuation:MOEXception
CONFigure:AFRF:MEASurement<Instance>:MEValuation:RCOupling
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TOUT
CONFigure:AFRF:MEASurement<Instance>:MEValuation:SAUTomatic
CONFigure:AFRF:MEASurement<Instance>:MEValuation:STFind
```

class MultiEvalCls

MultiEval commands group definition. 116 total commands, 13 Subgroups, 8 group commands

get_crepetition() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:CREPetition
value: bool = driver.configure.afRf.measurement.multiEval.get_crepetition()
```

Enables or disables the automatic configuration of the repetition mode. With enabled automatic configuration, the repetition mode of all measurements is set to 'Continuous' each time the instrument switches from remote operation to manual operation.

return
continuous_repetition: No help available

get_mo_exception() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:MOEXception
value: bool = driver.configure.afRf.measurement.multiEval.get_mo_exception()
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

return
meas_on_exception: OFF | ON OFF Faulty results are rejected ON Results are never rejected

get_rcoupling() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RCOupling
value: bool = driver.configure.afRf.measurement.multiEval.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupling: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:REPetition
value: enums.Repeat = driver.configure.afRf.measurement.multiEval.get_
↪repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

get_sautomatic() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SAUTomatic
value: bool = driver.configure.afRf.measurement.multiEval.get_sautomatic()
```

No command help available

return
start_meas_automatic: No help available

get_scondition() → StopCondition

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SCONdition
value: enums.StopCondition = driver.configure.afRf.measurement.multiEval.get_
↪scondition()
```

Selects whether the measurement is stopped after a failed limit check or continued.

return
stop_condition: NONE | SLFail NONE Continue measurement irrespective of the limit check SLFail Stop measurement on limit failure

get_st_find() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:STFind
value: bool = driver.configure.afRf.measurement.multiEval.get_st_find()
```

No command help available

return
start_find: No help available

get_timeout() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TOUT
value: float = driver.configure.afRf.measurement.multiEval.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_crepetition(*continuous_repetition: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:CREPetition
driver.configure.afRf.measurement.multiEval.set_crepetition(continuous_
↪repetition = False)
```

Enables or disables the automatic configuration of the repetition mode. With enabled automatic configuration, the repetition mode of all measurements is set to 'Continuous' each time the instrument switches from remote operation to manual operation.

param continuous_repetition
OFF | ON

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:MOEXception
driver.configure.afRf.measurement.multiEval.set_mo_exception(meas_on_exception␣
↪= False)
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

param meas_on_exception
OFF | ON OFF Faulty results are rejected ON Results are never rejected

set_rcoupling(*repetition_coupling: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RCOupling
driver.configure.afRf.measurement.multiEval.set_rcoupling(repetition_coupling =␣
↪False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupling
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:REPetition
driver.configure.afRf.measurement.multiEval.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

set_sautomatic(*start_meas_automatic: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SAUTomatic
driver.configure.afRf.measurement.multiEval.set_sautomatic(start_meas_automatic␣
↪= False)
```

No command help available

param start_meas_automatic
No help available

set_scondition(*stop_condition: StopCondition*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SCONdition
driver.configure.afRf.measurement.multiEval.set_scondition(stop_condition =␣
↪enums.StopCondition.NONE)
```

Selects whether the measurement is stopped after a failed limit check or continued.

param stop_condition
NONE | SLFail NONE Continue measurement irrespective of the limit check SLFail Stop measurement on limit failure

set_st_find(*start_find: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:STFind
driver.configure.afRf.measurement.multiEval.set_st_find(start_find = False)
```

No command help available

param start_find
No help available

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TOUT
driver.configure.afRf.measurement.multiEval.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.clone()
```

Subgroups

6.3.1.2.9.1 Af

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:AF:SCOunt
```

class AfCls

Af commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_scount() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:AF:SCOunt
value: int = driver.configure.afRf.measurement.multiEval.af.get_scount()
```

Specifies the number of measurement intervals per measurement cycle for AF results. One measurement interval delivers a single 'Current' value per result.

return
statistic_count: Range: 1 to 1000

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:AF:SCOunt
driver.configure.afRf.measurement.multiEval.af.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle for AF results. One measurement interval delivers a single 'Current' value per result.

param statistic_count
Range: 1 to 1000

6.3.1.2.9.2 AfFft

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:AFFFt:SCOunt
```

class AfFftCls

AfFft commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_scount() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:AFFFt:SCOunt
value: int = driver.configure.afRf.measurement.multiEval.afFft.get_scount()
```

Specifies the number of measurement intervals per measurement cycle for the AF spectrum diagram. One measurement interval delivers a single 'Current' trace.

return
statistic_count: Range: 1 to 1000

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:AFFFt:SCOunt
driver.configure.afRf.measurement.multiEval.afFft.set_scount(statistic_count =␣
↪1)
```

Specifies the number of measurement intervals per measurement cycle for the AF spectrum diagram. One measurement interval delivers a single 'Current' trace.

param statistic_count
Range: 1 to 1000

6.3.1.2.9.3 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.afRf.measurement.multiEval.audioInput.repcap_audioInput_get()
driver.configure.afRf.measurement.multiEval.audioInput.repcap_audioInput_set(repcap.
↪AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.audioInput.clone()
```

Subgroups

6.3.1.2.9.4 Tmode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:TMODe
```

class TmodeCls

Tmode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:TMODe
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪audioInput.tmode.get(audioInput = repcap.AudioInput.Default)
```

No command help available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
tone_mode: No help available

set(*tone_mode: DigitalToneMode*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:AIN<Nr>:TMODe
driver.configure.afRf.measurement.multiEval.audioInput.tmode.set(tone_mode =␣
↪enums.DigitalToneMode.DCS, audioInput = repcap.AudioInput.Default)
```

No command help available

param tone_mode
No help available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.5 Demodulation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:DEModulation:TMODe
```

class DemodulationCls

Demodulation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tmode() → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:DEModulation:TMODe
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪demodulation.get_tmode()
```

No command help available

return
tone_mode: No help available

set_tmode(*tone_mode: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:DEModulation:TMODe
driver.configure.afRf.measurement.multiEval.demodulation.set_tmode(tone_mode =␣
↪enums.DigitalToneMode.DCS)
```

No command help available

param tone_mode
No help available

6.3.1.2.9.6 Fft

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:SPAN
CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:LENGth
CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:WINDow
```

class FftCls

Fft commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_length() → FftLength

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:LENGth
value: enums.FftLength = driver.configure.afRf.measurement.multiEval.fft.get_
↪length()
```

No command help available

return
length: No help available

get_span() → FftSpan

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:SPAN
value: enums.FftSpan = driver.configure.afRf.measurement.multiEval.fft.get_
↪span()
```

No command help available

return
span: No help available

get_window() → FftWindowType

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:WINDow
value: enums.FftWindowType = driver.configure.afRf.measurement.multiEval.fft.
↪get_window()
```

Selects the window function to be applied before the fast Fourier transformation.

return
type_py: RECTangle | HAMMing | HANN | BLHA | FLTP RECTangle, HAMMing, HANN Rectangular / Hamming / Hann window BLHA Blackman-Harris window FLTP Flat-Top window

set_length(*length: FftLength*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:LENGth
driver.configure.afRf.measurement.multiEval.fft.set_length(length = enums.
↪FftLength.F16K)
```

No command help available

param length
No help available

set_span(*span: FftSpan*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:SPAN
driver.configure.afRf.measurement.multiEval.fft.set_span(span = enums.FftSpan.
↪SP1)
```

No command help available

param span
No help available

set_window(*type_py: FftWindowType*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:WINDow
driver.configure.afRf.measurement.multiEval.fft.set_window(type_py = enums.
↪FftWindowType.BLHA)
```

Selects the window function to be applied before the fast Fourier transformation.

param type_py
RECTangle | HAMMing | HANN | BLHA | FLTP RECTangle, HAMMing, HANN Rectangular / Hamming / Hann window BLHA Blackman-Harris window FLTP Flat-Top window

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.fft.clone()
```

Subgroups

6.3.1.2.9.7 Marker<MarkerOther>

RepCap Settings

```
# Range: Nr2 .. Nr5
rc = driver.configure.afRf.measurement.multiEval.fft.marker.repcap_markerOther_get()
driver.configure.afRf.measurement.multiEval.fft.marker.repcap_markerOther_set(repcap.
↪MarkerOther.Nr2)
```

class MarkerCls

Marker commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: MarkerOther, default value after init: MarkerOther.Nr2

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.fft.marker.clone()
```

Subgroups

6.3.1.2.9.8 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:MARKer:ENABle:ALL
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set_all(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:MARKer:ENABle:ALL
driver.configure.afRf.measurement.multiEval.fft.marker.enable.set_all(enable =␣
↪False)
```

Enables or disables the markers R, 2 and 3.

param enable
OFF | ON

6.3.1.2.9.9 Placement

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:MARKer<mnr>:PLACement
```

class PlacementCls

Placement commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*markerOther=MarkerOther.Default*) → MarkerPlacement

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:MARKer<mnr>
↪:PLACement
value: enums.MarkerPlacement = driver.configure.afRf.measurement.multiEval.fft.
↪marker.placement.get(markerOther = repcap.MarkerOther.Default)
```

Selects between absolute coordinates and delta coordinates relative to the reference marker, for marker number <mnr>.

param markerOther
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Marker')

return
placement: ABSolute | RELative

set(*placement: MarkerPlacement*, *markerOther=MarkerOther.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FFT:MARKer<mnr>
↪:PLACement
driver.configure.afRf.measurement.multiEval.fft.marker.placement.set(placement␣
↪= enums.MarkerPlacement.ABSolute, markerOther = repcap.MarkerOther.Default)
```

Selects between absolute coordinates and delta coordinates relative to the reference marker, for marker number <mnr>.

param placement
ABSolute | RELative

param markerOther
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Marker')

6.3.1.2.9.10 FilterPy

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:FILTer:BWIDth
```

class FilterPyCls

FilterPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_bandwidth() → BandpassFilter

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FILTer:BWIDth
value: enums.BandpassFilter = driver.configure.afRf.measurement.multiEval.
↪filterPy.get_bandwidth()
```

Configures the bandwidth of the bandpass filter in the RF input path.

return
bandpass: F8330 | F25K | F50K | F01M | F05M Bandwidth 8330 Hz, 25 kHz, 50 kHz, 0.1 MHz, 0.5 MHz

set_bandwidth(*bandpass: BandpassFilter*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:FILTer:BWIDth
driver.configure.afRf.measurement.multiEval.filterPy.set_bandwidth(bandpass =␣
↪enums.BandpassFilter.F01M)
```

Configures the bandwidth of the bandpass filter in the RF input path.

param bandpass
F8330 | F25K | F50K | F01M | F05M Bandwidth 8330 Hz, 25 kHz, 50 kHz, 0.1 MHz, 0.5 MHz

6.3.1.2.9.11 Limit

class LimitCls

Limit commands group definition. 46 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.clone()
```

Subgroups

6.3.1.2.9.12 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.afRf.measurement.multiEval.limit.audioInput.repcap_audioInput_get()
driver.configure.afRf.measurement.multiEval.limit.audioInput.repcap_audioInput_
↪set(repcap.AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 6 total commands, 6 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.audioInput.clone()
```

Subgroups

6.3.1.2.9.13 Sinad

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SINad
```

class SinadCls

Sinad commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SinadStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower SINAD limit Range: 0 dB to 140 dB, Unit: dB
- Upper: float: Upper SINAD limit Range: 0 dB to 140 dB, Unit: dB

get(*audioInput=AudioInput.Default*) → SinadStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SINad
value: SinadStruct = driver.configure.afRf.measurement.multiEval.limit.
↪audioInput.sinad.get(audioInput = repcap.AudioInput.Default)
```

Configures limits for the SINAD results, measured via an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for SinadStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float = None*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SINad
driver.configure.afRf.measurement.multiEval.limit.audioInput.sinad.set(enable =␣
↪False, lower = 1.0, upper = 1.0, audioInput = repcap.AudioInput.Default)
```

Configures limits for the SINAD results, measured via an AF input path.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower SINAD limit Range: 0 dB to 140 dB, Unit: dB

param upper
Upper SINAD limit Range: 0 dB to 140 dB, Unit: dB

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.14 SndRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNDRatio
```

class SndRatioCls

SndRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SndRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get(*audioInput=AudioInput.Default*) → SndRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNDRatio
value: SndRatioStruct = driver.configure.afRf.measurement.multiEval.limit.
↪audioInput.sndRatio.get(audioInput = repcap.AudioInput.Default)
```

Configures limits for all SNR results, measured via an AF input path. SNR results (signal/noise) include S/N, (S+N) /N and (S+N+D) /N.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for SndRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float = None*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNDRatio
driver.configure.afRf.measurement.multiEval.limit.audioInput.sndRatio.
↪set(enable = False, lower = 1.0, upper = 1.0, audioInput = repcap.AudioInput.
↪Default)
```

Configures limits for all SNR results, measured via an AF input path. SNR results (signal/noise) include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.15 SnnRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNNRatio
```

class SnnRatioCls

SnnRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SnnRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get(*audioInput=AudioInput.Default*) → SnnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNNRatio
value: SnnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.
↪audioInput.snnRatio.get(audioInput = repcap.AudioInput.Default)
```

Configures limits for all SNR results, measured via an AF input path. SNR results (signal/noise) include S/N, (S+N) /N and (S+N+D) /N.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for SnnRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float = None*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNNRatio
driver.configure.afRf.measurement.multiEval.limit.audioInput.snnRatio.
↪set(enable = False, lower = 1.0, upper = 1.0, audioInput = repcap.AudioInput.
↪Default)
```

Configures limits for all SNR results, measured via an AF input path. SNR results (signal/noise) include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.16 SnRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNRatio
```

class SnRatioCls

SnRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SnRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get(*audioInput=AudioInput.Default*) → SnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNRatio
value: SnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.
↪audioInput.snRatio.get(audioInput = repcap.AudioInput.Default)
```

Configures limits for all SNR results, measured via an AF input path. SNR results (signal/noise) include S/N, (S+N) /N and (S+N+D) /N.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for SnRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float = None*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:SNRatio
driver.configure.afRf.measurement.multiEval.limit.audioInput.snRatio.set(enable␣
↪= False, lower = 1.0, upper = 1.0, audioInput = repcap.AudioInput.Default)
```

Configures limits for all SNR results, measured via an AF input path. SNR results (signal/noise) include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.17 ThDistortion

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:THDistortion
```

class ThDistortionCls

ThDistortion commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThDistortionStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper THD limit Range: 0 % to 100 %, Unit: %

get(*audioInput=AudioInput.Default*) → ThDistortionStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>
↪:THDistortion
value: ThDistortionStruct = driver.configure.afRf.measurement.multiEval.limit.
↪audioInput.thDistortion.get(audioInput = repcap.AudioInput.Default)
```

Configures a limit for the THD results, measured via an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ThDistortionStruct structure arguments.

set(*enable: bool*, *upper: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>
↪:THDistortion
driver.configure.afRf.measurement.multiEval.limit.audioInput.thDistortion.
↪set(enable = False, upper = 1.0, audioInput = repcap.AudioInput.Default)
```

Configures a limit for the THD results, measured via an AF input path.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper THD limit Range: 0 % to 100 %, Unit: %

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.18 ThdNoise

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:THDNoise
```

class ThdNoiseCls

ThdNoise commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThdNoiseStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper THD+N limit Range: 0.001 % to 100 %, Unit: %

get(*audioInput=AudioInput.Default*) → ThdNoiseStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:THDNoise
value: ThdNoiseStruct = driver.configure.afRf.measurement.multiEval.limit.
↪audioInput.thdNoise.get(audioInput = repcap.AudioInput.Default)
```

Configures a limit for the THD+N results, measured via an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for ThdNoiseStruct structure arguments.

set(*enable: bool*, *upper: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:AIN<Nr>:THDNoise
driver.configure.afRf.measurement.multiEval.limit.audioInput.thdNoise.
↪set(enable = False, upper = 1.0, audioInput = repcap.AudioInput.Default)
```

Configures a limit for the THD+N results, measured via an AF input path.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper THD+N limit Range: 0.001 % to 100 %, Unit: %

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.19 Demodulation

class DemodulationCls

Demodulation commands group definition. 16 total commands, 9 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.demodulation.clone()
```

Subgroups

6.3.1.2.9.20 Fdeviation

class FdeviationCls

Fdeviation commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.demodulation.fdeviation.
↪clone()
```

Subgroups

6.3.1.2.9.21 Peak

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:FDEViation:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: A query returns the lower limit. A setting applies the absolute value to the upper limit and the absolute value plus a negative sign to the lower limit. Range: -100 kHz to 0 Hz, Unit: Hz
- Upper: float: A query returns the upper limit. A setting ignores this parameter. Range: 0 Hz to 100 kHz, Unit: Hz

get() → PeakStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FDEViation:PEAK
value: PeakStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fdeviation.peak.get()
```

Configures limits for the frequency deviation results '+Peak' and '-Peak', measured for FM. The upper and lower limits have the same absolute value but different signs.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FDEViation:PEAK
driver.configure.afRf.measurement.multiEval.limit.demodulation.fdeviation.peak.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for the frequency deviation results '+Peak' and '-Peak', measured for FM. The upper and lower limits have the same absolute value but different signs.

param enable
OFF | ON Enables or disables the limit check

param lower
A query returns the lower limit. A setting applies the absolute value to the upper limit and the absolute value plus a negative sign to the lower limit. Range: -100 kHz to 0 Hz, Unit: Hz

param upper
A query returns the upper limit. A setting ignores this parameter. Range: 0 Hz to 100 kHz, Unit: Hz

6.3.1.2.9.22 Rms

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:FDEViation:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper frequency deviation limit Range: 0 Hz to 100 kHz, Unit: Hz

get() → RmsStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FDEViation:RMS
value: RmsStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fdeviation.rms.get()
```

Configures a limit for the 'RMS' frequency deviation results, measured for FM.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FDEViation:RMS
driver.configure.afRf.measurement.multiEval.limit.demodulation.fdeviation.rms.
↪set(enable = False, upper = 1.0)
```

Configures a limit for the 'RMS' frequency deviation results, measured for FM.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper frequency deviation limit Range: 0 Hz to 100 kHz, Unit: Hz

6.3.1.2.9.23 FmStereo

class FmStereoCls

FmStereo commands group definition. 6 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.clone()
```

Subgroups

6.3.1.2.9.24 Adeviation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:FMSTereo:ADEViation
```

class AdeviationCls

Adeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class AdeviationStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper audio deviation limit Range: 0 Hz to 100 kHz, Unit: Hz

get() → AdeviationStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:ADEViation
value: AdeviationStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fmStereo.adeviation.get()
```

Configures a limit for the audio deviation results, measured for FM stereo.

return
structure: for return value, see the help for AdeviationStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:ADEViation
driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.
↪adeviation.set(enable = False, upper = 1.0)
```

Configures a limit for the audio deviation results, measured for FM stereo.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper audio deviation limit Range: 0 Hz to 100 kHz, Unit: Hz

6.3.1.2.9.25 Mdeviation

class MdeviationCls

Mdeviation commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.
↪mdeviation.clone()
```

Subgroups

6.3.1.2.9.26 Peak

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:MDEViation:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check

- Lower: float: Lower multiplex deviation limit A setting configures also the upper limit (same absolute value, positive sign) . Range: -100 kHz to 0 Hz, Unit: Hz
- Upper: float: A query returns the upper multiplex deviation limit. A setting ignores this parameter. Range: 0 Hz to 100 kHz, Unit: Hz

get() → PeakStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:MDEViation:PEAK
value: PeakStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fmStereo.mdeviation.peak.get()
```

Configures limits for the multiplex deviation results '+Peak' and '-Peak', measured for FM stereo.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:MDEViation:PEAK
driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.
↪mdeviation.peak.set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for the multiplex deviation results '+Peak' and '-Peak', measured for FM stereo.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower multiplex deviation limit A setting configures also the upper limit (same absolute value, positive sign) . Range: -100 kHz to 0 Hz, Unit: Hz

param upper
A query returns the upper multiplex deviation limit. A setting ignores this parameter. Range: 0 Hz to 100 kHz, Unit: Hz

6.3.1.2.9.27 Rms

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:MDEViation:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper multiplex deviation limit Range: 0 Hz to 100 kHz, Unit: Hz

get() → RmsStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:MDEViation:RMS
value: RmsStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fmStereo.mdeviation.rms.get()
```

Configures a limit for the 'RMS' multiplex deviation results, measured for FM stereo.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:MDEViation:RMS
driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.
↪mdeviation.rms.set(enable = False, upper = 1.0)
```

Configures a limit for the 'RMS' multiplex deviation results, measured for FM stereo.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper multiplex deviation limit Range: 0 Hz to 100 kHz, Unit: Hz

6.3.1.2.9.28 PfError

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:FMSTereo:PFERror
```

class PfErrorCls

PfError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PfErrorStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower frequency error limit Range: -100 Hz to 0 Hz, Unit: Hz
- Upper: float: Upper frequency error limit You can skip this setting. The Upper value equals always the Lower value times -1 (same absolute value, opposite sign) . Range: 0 Hz to 100 Hz, Unit: Hz

get() → PfErrorStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:PFERror
value: PfErrorStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fmStereo.pfError.get()
```

Configures limits for the pilot frequency error, measured for FM stereo.

return
structure: for return value, see the help for PfErrorStruct structure arguments.

set(*enable: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:PFERror
driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.pfError.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for the pilot frequency error, measured for FM stereo.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower frequency error limit Range: -100 Hz to 0 Hz, Unit: Hz

param upper
Upper frequency error limit You can skip this setting. The Upper value equals always the Lower value times -1 (same absolute value, opposite sign) . Range: 0 Hz to 100 Hz, Unit: Hz

6.3.1.2.9.29 PiDeviation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:FMSTereo:PIDeviation
```

class PiDeviationCls

PiDeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PiDeviationStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper pilot deviation limit Range: 0 Hz to 10 kHz, Unit: Hz

get() → PiDeviationStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:PIDeviation
value: PiDeviationStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fmStereo.piDeviation.get()
```

Configures a limit for the pilot deviation results, measured for FM stereo.

return
structure: for return value, see the help for PiDeviationStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:PIDeviation
driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.
↪piDeviation.set(enable = False, upper = 1.0)
```

Configures a limit for the pilot deviation results, measured for FM stereo.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper pilot deviation limit Range: 0 Hz to 10 kHz, Unit: Hz

6.3.1.2.9.30 RdsDeviation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:FMSTereo:RDSDeviation
```

class RdsDeviationCls

RdsDeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RdsDeviationStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper RDS deviation limit Range: 0 Hz to 10 kHz, Unit: Hz

get() → RdsDeviationStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:RDSDeviation
value: RdsDeviationStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.fmStereo.rdsDeviation.get()
```

Configures a limit for the RDS deviation, measured for FM stereo.

return
structure: for return value, see the help for RdsDeviationStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:FMSTereo:RDSDeviation
driver.configure.afRf.measurement.multiEval.limit.demodulation.fmStereo.
↪rdsDeviation.set(enable = False, upper = 1.0)
```

Configures a limit for the RDS deviation, measured for FM stereo.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper RDS deviation limit Range: 0 Hz to 10 kHz, Unit: Hz

6.3.1.2.9.31 Pdeviation

class PdeviationCls

Pdeviation commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.demodulation.pdeviation.
↪clone()
```

Subgroups

6.3.1.2.9.32 Peak

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:PDEViation:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: A query returns the lower limit. A setting applies the absolute value to the upper limit and the absolute value plus a negative sign to the lower limit. Range: -10 rad to 0 rad, Unit: rad
- Upper: float: A query returns the upper limit. A setting ignores this parameter. Range: 0 rad to 10 rad, Unit: rad

get() → PeakStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:PDEViation:PEAK
value: PeakStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.pdeviation.peak.get()
```

Configures limits for the phase deviation results '+Peak' and '-Peak', measured for PM. The upper and lower limits have the same absolute value but different signs.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:PDEViation:PEAK
driver.configure.afRf.measurement.multiEval.limit.demodulation.pdeviation.peak.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for the phase deviation results '+Peak' and '-Peak', measured for PM. The upper and lower limits have the same absolute value but different signs.

param enable
OFF | ON Enables or disables the limit check

param lower
A query returns the lower limit. A setting applies the absolute value to the upper limit and the absolute value plus a negative sign to the lower limit. Range: -10 rad to 0 rad, Unit: rad

param upper
A query returns the upper limit. A setting ignores this parameter. Range: 0 rad to 10 rad, Unit: rad

6.3.1.2.9.33 Rms

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:PDEViation:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper phase deviation limit Range: 0 rad to 10 rad, Unit: rad

get() → RmsStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:PDEViation:RMS
value: RmsStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.pdeviation.rms.get()
```

Configures a limit for the 'RMS' phase deviation results, measured for PM.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:PDEViation:RMS
driver.configure.afRf.measurement.multiEval.limit.demodulation.pdeviation.rms.
↪set(enable = False, upper = 1.0)
```

Configures a limit for the 'RMS' phase deviation results, measured for PM.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper phase deviation limit Range: 0 rad to 10 rad, Unit: rad

6.3.1.2.9.34 Sinad

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:SINad
```

class SinadCls

Sinad commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SinadStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower SINAD limit Range: 0 dB to 140 dB, Unit: dB
- Upper: float: Upper SINAD limit Range: 0 dB to 140 dB, Unit: dB

get() → SinadStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SINad
value: SinadStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.sinad.get()
```

Configures limits for the SINAD results, measured via the RF input path.

return
structure: for return value, see the help for SinadStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float* = *None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SINad
driver.configure.afRf.measurement.multiEval.limit.demodulation.sinad.set(enable␣
↪= False, lower = 1.0, upper = 1.0)
```

Configures limits for the SINAD results, measured via the RF input path.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower SINAD limit Range: 0 dB to 140 dB, Unit: dB

param upper
Upper SINAD limit Range: 0 dB to 140 dB, Unit: dB

6.3.1.2.9.35 SndRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:SNDRatio
```

class SndRatioCls

SndRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SndRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get() → SndRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SNDRatio
value: SndRatioStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.sndRatio.get()
```

Configures limits for all SNR results, measured via the RF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SndRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float* = *None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SNDRatio
driver.configure.afRf.measurement.multiEval.limit.demodulation.sndRatio.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for all SNR results, measured via the RF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

6.3.1.2.9.36 SnnRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:SNNRatio
```

class SnnRatioCls

SnnRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SnnRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get() → SnnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SNNRatio
value: SnnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.snnRatio.get()
```

Configures limits for all SNR results, measured via the RF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SnnRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float* = *None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SNNRatio
driver.configure.afRf.measurement.multiEval.limit.demodulation.snnRatio.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for all SNR results, measured via the RF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

6.3.1.2.9.37 SnRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:SNRatio
```

class SnRatioCls

SnRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SnRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get() → SnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SNRatio
value: SnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.snRatio.get()
```

Configures limits for all SNR results, measured via the RF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SnRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float* = *None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:SNRatio
driver.configure.afRf.measurement.multiEval.limit.demodulation.snRatio.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for all SNR results, measured via the RF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

6.3.1.2.9.38 ThDistortion

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:THDistortion
```

class ThDistortionCls

ThDistortion commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThDistortionStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper THD limit Range: 0.001 % to 100 %, Unit: %

get() → ThDistortionStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:THDistortion
value: ThDistortionStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.thDistortion.get()
```

Configures a limit for the THD results, measured via the RF input path.

return
structure: for return value, see the help for ThDistortionStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:THDistortion
driver.configure.afRf.measurement.multiEval.limit.demodulation.thDistortion.
↪set(enable = False, upper = 1.0)
```

Configures a limit for the THD results, measured via the RF input path.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper THD limit Range: 0.001 % to 100 %, Unit: %

6.3.1.2.9.39 ThdNoise

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:DEModulation:THDNoise
```

class ThdNoiseCls

ThdNoise commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThdNoiseStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check

- Upper: float: Upper THD+N limit Range: 0.001 % to 100 %, Unit: %

get() → ThdNoiseStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:THDNoise
value: ThdNoiseStruct = driver.configure.afRf.measurement.multiEval.limit.
↪demodulation.thdNoise.get()
```

Configures a limit for the THD+N results, measured via the RF input path.

return
structure: for return value, see the help for ThdNoiseStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:DEModulation:THDNoise
driver.configure.afRf.measurement.multiEval.limit.demodulation.thdNoise.
↪set(enable = False, upper = 1.0)
```

Configures a limit for the THD+N results, measured via the RF input path.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper THD+N limit Range: 0.001 % to 100 %, Unit: %

6.3.1.2.9.40 RfCarrier

class RfCarrierCls

RfCarrier commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.rfCarrier.clone()
```

Subgroups

6.3.1.2.9.41 FreqError

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:FERRor
```

class FreqErrorCls

FreqError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FreqErrorStruct

Response structure. Fields:

- Enable_Limit: bool: OFF | ON Enables or disables the limit check
- Lower: float: A query returns the lower limit. A setting applies the absolute value to the upper limit and the absolute value plus a negative sign to the lower limit. Range: -50 kHz to 0 Hz, Unit: Hz
- Upper: float: A query returns the upper limit. A setting ignores this parameter. Range: 0 Hz to 50 kHz, Unit: Hz

get() → FreqErrorStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:FERRor
value: FreqErrorStruct = driver.configure.afRf.measurement.multiEval.limit.
↪rfCarrier.freqError.get()
```

Configures limits for the measured RF carrier frequency error. The upper and lower limits have the same absolute value but different signs.

return
structure: for return value, see the help for FreqErrorStruct structure arguments.

set(*enable_limit: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:FERRor
driver.configure.afRf.measurement.multiEval.limit.rfCarrier.freqError.
↪set(enable_limit = False, lower = 1.0, upper = 1.0)
```

Configures limits for the measured RF carrier frequency error. The upper and lower limits have the same absolute value but different signs.

param enable_limit
OFF | ON Enables or disables the limit check

param lower
A query returns the lower limit. A setting applies the absolute value to the upper limit and the absolute value plus a negative sign to the lower limit. Range: -50 kHz to 0 Hz, Unit: Hz

param upper
A query returns the upper limit. A setting ignores this parameter. Range: 0 Hz to 50 kHz, Unit: Hz

6.3.1.2.9.42 PePower

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:PEPower
```

class PePowerCls

PePower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PePowerStruct

Response structure. Fields:

- Enable_Limit: bool: OFF | ON Enables or disables the limit check

- Lower: float: Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm
- Upper: float: Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

get() → PePowerStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:PEPower
value: PePowerStruct = driver.configure.afRf.measurement.multiEval.limit.
↪rfCarrier.pePower.get()
```

Configures limits for the measured RF signal power (PEP value) .

return
structure: for return value, see the help for PePowerStruct structure arguments.

set(*enable_limit: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:PEPower
driver.configure.afRf.measurement.multiEval.limit.rfCarrier.pePower.set(enable_
↪limit = False, lower = 1.0, upper = 1.0)
```

Configures limits for the measured RF signal power (PEP value) .

param enable_limit
OFF | ON Enables or disables the limit check

param lower
Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm

param upper
Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

6.3.1.2.9.43 Power

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:POWer
```

class PowerCls

Power commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PowerStruct

Response structure. Fields:

- Enable_Limit: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm
- Upper: float: Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

get() → PowerStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:POWer
value: PowerStruct = driver.configure.afRf.measurement.multiEval.limit.
↪rfCarrier.power.get()
```

Configures limits for the measured RF signal power (RMS value) .

return
structure: for return value, see the help for PowerStruct structure arguments.

`set`(*enable_limit: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:RFCarrier:POWer
driver.configure.afRf.measurement.multiEval.limit.rfCarrier.power.set(enable_
↪limit = False, lower = 1.0, upper = 1.0)
```

Configures limits for the measured RF signal power (RMS value) .

param enable_limit
OFF | ON Enables or disables the limit check

param lower
Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm

param upper
Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

6.3.1.2.9.44 Spdif

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNRatio
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SINad
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNNRatio
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNDRatio
```

`class SpdifCls`

Spdif commands group definition. 6 total commands, 2 Subgroups, 4 group commands

`class SinadStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Enable_Left: bool: OFF | ON Enables or disables the limit check for the left SPDIF channel
- Lower_Left: float: Lower SINAD limit for the left SPDIF channel Range: 0 dB to 140 dB, Unit: dB
- Enable_Right: bool: OFF | ON Enables or disables the limit check for the right SPDIF channel
- Lower_Right: float: Lower SINAD limit for the right SPDIF channel Range: 0 dB to 140 dB, Unit: dB
- Upper_Left: float: Optional setting parameter. Upper SINAD limit for the left SPDIF channel Range: 0 dB to 140 dB, Unit: dB
- Upper_Right: float: Optional setting parameter. Upper SINAD limit for the right SPDIF channel Range: 0 dB to 140 dB, Unit: dB

`class SnRatioStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Enable_Left: bool: OFF | ON Enables or disables the limit check for the left SPDIF channel.
- Lower_Left: float: Lower limit for the left SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Enable_Right: bool: OFF | ON Enables or disables the limit check for the right SPDIF channel.
- Lower_Right: float: Lower limit for the right SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB

- Upper_Left: float: Optional setting parameter. Upper limit for the left SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper_Right: float: Optional setting parameter. Upper limit for the right SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB

class SndRatioStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Enable_Left: bool: OFF | ON Enables or disables the limit check for the left SPDIF channel.
- Lower_Left: float: Lower limit for the left SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Enable_Right: bool: OFF | ON Enables or disables the limit check for the right SPDIF channel.
- Lower_Right: float: Lower limit for the right SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper_Left: float: Optional setting parameter. Upper limit for the left SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper_Right: float: Optional setting parameter. Upper limit for the right SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB

class SnnRatioStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Enable_Left: bool: OFF | ON Enables or disables the limit check for the left SPDIF channel.
- Lower_Left: float: Lower limit for the left SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Enable_Right: bool: OFF | ON Enables or disables the limit check for the right SPDIF channel.
- Lower_Right: float: Lower limit for the right SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper_Left: float: Optional setting parameter. Upper limit for the left SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper_Right: float: Optional setting parameter. Upper limit for the right SPDIF channel Range: 0.00 dB to 140.00 dB, Unit: dB

get_sinad() → SinadStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SINad
value: SinadStruct = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪get_sinad()
```

Configures limits for the SINAD results, measured via the SPDIF input path.

return
structure: for return value, see the help for SinadStruct structure arguments.

get_sn_ratio() → SnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNRatio
value: SnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪get_sn_ratio()
```

Configures limits for all SNR results, measured via the SPDIF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SnRatioStruct structure arguments.

get_snd_ratio() → SndRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNDRatio
value: SndRatioStruct = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪get_snd_ratio()
```

Configures limits for all SNR results, measured via the SPDIF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SndRatioStruct structure arguments.

get_snn_ratio() → SnnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNNRatio
value: SnnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪get_snn_ratio()
```

Configures limits for all SNR results, measured via the SPDIF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SnnRatioStruct structure arguments.

set_sinad(*value: SinadStruct*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SINad
structure = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪SinadStruct()
structure.Enable_Left: bool = False
structure.Lower_Left: float = 1.0
structure.Enable_Right: bool = False
structure.Lower_Right: float = 1.0
structure.Upper_Left: float = 1.0
structure.Upper_Right: float = 1.0
driver.configure.afRf.measurement.multiEval.limit.spdif.set_sinad(value =␣
↪structure)
```

Configures limits for the SINAD results, measured via the SPDIF input path.

param value
see the help for SinadStruct structure arguments.

set_sn_ratio(*value: SnRatioStruct*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNRatio
structure = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪SnRatioStruct()
structure.Enable_Left: bool = False
structure.Lower_Left: float = 1.0
structure.Enable_Right: bool = False
structure.Lower_Right: float = 1.0
structure.Upper_Left: float = 1.0
structure.Upper_Right: float = 1.0
driver.configure.afRf.measurement.multiEval.limit.spdif.set_sn_ratio(value =␣
↪structure)
```

Configures limits for all SNR results, measured via the SPDIF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param value
see the help for SnRatioStruct structure arguments.

set_snd_ratio(*value: SndRatioStruct*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNDRatio
structure = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪SndRatioStruct()
structure.Enable_Left: bool = False
structure.Lower_Left: float = 1.0
structure.Enable_Right: bool = False
structure.Lower_Right: float = 1.0
structure.Upper_Left: float = 1.0
structure.Upper_Right: float = 1.0
driver.configure.afRf.measurement.multiEval.limit.spdif.set_snd_ratio(value =␣
↪structure)
```

Configures limits for all SNR results, measured via the SPDIF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param value
see the help for SndRatioStruct structure arguments.

set_snn_ratio(*value: SnnRatioStruct*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:SNNRatio
structure = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪SnnRatioStruct()
structure.Enable_Left: bool = False
structure.Lower_Left: float = 1.0
structure.Enable_Right: bool = False
structure.Lower_Right: float = 1.0
structure.Upper_Left: float = 1.0
structure.Upper_Right: float = 1.0
driver.configure.afRf.measurement.multiEval.limit.spdif.set_snn_ratio(value =␣
↪structure)
```

Configures limits for all SNR results, measured via the SPDIF input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param value
see the help for SnnRatioStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.spdif.clone()
```

Subgroups

6.3.1.2.9.45 ThDistortion

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:THDistortion
```

class ThDistortionCls

ThDistortion commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThDistortionStruct

Response structure. Fields:

- Enable_Left: bool: OFF | ON Enables or disables the limit check for the left SPDIF channel
- Upper_Left: float: Upper THD limit for the left SPDIF channel Range: 0 % to 100 %, Unit: %
- Enable_Right: bool: OFF | ON Enables or disables the limit check for the right SPDIF channel
- Upper_Right: float: Upper THD limit for the right SPDIF channel Range: 0 % to 100 %, Unit: %

get() → ThDistortionStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:THDistortion
value: ThDistortionStruct = driver.configure.afRf.measurement.multiEval.limit.
↪spdif.thDistortion.get()
```

Configures limits for the THD results, measured via the SPDIF input path.

return
structure: for return value, see the help for ThDistortionStruct structure arguments.

set(*enable_left: bool*, *upper_left: float*, *enable_right: bool*, *upper_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:THDistortion
driver.configure.afRf.measurement.multiEval.limit.spdif.thDistortion.set(enable_
↪left = False, upper_left = 1.0, enable_right = False, upper_right = 1.0)
```

Configures limits for the THD results, measured via the SPDIF input path.

param enable_left
OFF | ON Enables or disables the limit check for the left SPDIF channel

param upper_left
Upper THD limit for the left SPDIF channel Range: 0 % to 100 %, Unit: %

param enable_right
OFF | ON Enables or disables the limit check for the right SPDIF channel

param upper_right
Upper THD limit for the right SPDIF channel Range: 0 % to 100 %, Unit: %

6.3.1.2.9.46 ThdNoise

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:THDNoise
```

class ThdNoiseCls

ThdNoise commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThdNoiseStruct

Response structure. Fields:

- Enable_Left: bool: OFF | ON Enables or disables the limit check for the left SPDIF channel
- Upper_Left: float: Upper THD+N limit for the left SPDIF channel Range: 0.001 % to 100 %, Unit: %
- Enable_Right: bool: OFF | ON Enables or disables the limit check for the right SPDIF channel
- Upper_Right: float: Upper THD+N limit for the right SPDIF channel Range: 0.001 % to 100 %, Unit: %

get() → ThdNoiseStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:THDNoise
value: ThdNoiseStruct = driver.configure.afRf.measurement.multiEval.limit.spdif.
↪thdNoise.get()
```

Configures limits for the THD+N results, measured via the SPDIF input path.

return
structure: for return value, see the help for ThdNoiseStruct structure arguments.

set(*enable_left: bool*, *upper_left: float*, *enable_right: bool*, *upper_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:SIN:THDNoise
driver.configure.afRf.measurement.multiEval.limit.spdif.thdNoise.set(enable_
↪left = False, upper_left = 1.0, enable_right = False, upper_right = 1.0)
```

Configures limits for the THD+N results, measured via the SPDIF input path.

param enable_left
OFF | ON Enables or disables the limit check for the left SPDIF channel

param upper_left
Upper THD+N limit for the left SPDIF channel Range: 0.001 % to 100 %, Unit: %

param enable_right
OFF | ON Enables or disables the limit check for the right SPDIF channel

param upper_right
Upper THD+N limit for the right SPDIF channel Range: 0.001 % to 100 %, Unit: %

6.3.1.2.9.47 Tones

class TonesCls

Tones commands group definition. 9 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.tones.clone()
```

Subgroups

6.3.1.2.9.48 Dcs

class DcsCls

Dcs commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.tones.dcs.clone()
```

Subgroups

6.3.1.2.9.49 FskDeviation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DCS:FSKDeviation
```

class FskDeviationCls

FskDeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FskDeviationStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower limit Range: -10 kHz to 0 Hz, Unit: Hz
- Upper: float: Upper limit Range: 0 Hz to 10 kHz, Unit: Hz

get() → FskDeviationStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:DCS:FSKDeviation
value: FskDeviationStruct = driver.configure.afRf.measurement.multiEval.limit.
↪tones.dcs.fskDeviation.get()
```

Configures limits for the FSK deviation measured for a DCS signal.

return
structure: for return value, see the help for FskDeviationStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:DCS:FSKDeviation
driver.configure.afRf.measurement.multiEval.limit.tones.dcs.fskDeviation.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for the FSK deviation measured for a DCS signal.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower limit Range: -10 kHz to 0 Hz, Unit: Hz

param upper
Upper limit Range: 0 Hz to 10 kHz, Unit: Hz

6.3.1.2.9.50 TocLength

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DCS:TOCLength
```

class TocLengthCls

TocLength commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TocLengthStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower limit Range: 0 s to 0.15 s, Unit: s
- Upper: float: Upper limit Range: 0.15 s to 1 s, Unit: s

get() → TocLengthStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:DCS:TOCLength
value: TocLengthStruct = driver.configure.afRf.measurement.multiEval.limit.
↪tones.dcs.tocLength.get()
```

Configures limits for the duration of DCS turn-off code transmissions.

return
structure: for return value, see the help for TocLengthStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:DCS:TOCLength
driver.configure.afRf.measurement.multiEval.limit.tones.dcs.tocLength.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for the duration of DCS turn-off code transmissions.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower limit Range: 0 s to 0.15 s, Unit: s

param upper
Upper limit Range: 0.15 s to 1 s, Unit: s

6.3.1.2.9.51 TofDeviation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DCS:TOFDeviation
```

class TofDeviationCls

TofDeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TofDeviationStruct

Response structure. Fields:

- Enable: bool: No parameter help available
- Upper: float: No parameter help available

get() → TofDeviationStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:DCS:TOFDeviation
value: TofDeviationStruct = driver.configure.afRf.measurement.multiEval.limit.
↪tones.dcs.tofDeviation.get()
```

No command help available

return
structure: for return value, see the help for TofDeviationStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:DCS:TOFDeviation
driver.configure.afRf.measurement.multiEval.limit.tones.dcs.tofDeviation.
↪set(enable = False, upper = 1.0)
```

No command help available

param enable
No help available

param upper
No help available

6.3.1.2.9.52 DigPause

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DIGPause
```

class DigPauseCls

DigPause commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DigPauseStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower pause limit Range: -100 % to 0 %, Unit: %
- Upper: float: Upper pause limit Range: 0 % to 100 %, Unit: %

get() → DigPauseStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DIGPause
value: DigPauseStruct = driver.configure.afRf.measurement.multiEval.limit.tones.
↪digPause.get()
```

Configures limits for the pause between two tones of an analyzed tone sequence (DTMF, free dialing and SelCall) .

return
structure: for return value, see the help for DigPauseStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DIGPause
driver.configure.afRf.measurement.multiEval.limit.tones.digPause.set(enable =␣
↪False, lower = 1.0, upper = 1.0)
```

Configures limits for the pause between two tones of an analyzed tone sequence (DTMF, free dialing and SelCall) .

param enable
OFF | ON Enables or disables the limit check

param lower
Lower pause limit Range: -100 % to 0 %, Unit: %

param upper
Upper pause limit Range: 0 % to 100 %, Unit: %

6.3.1.2.9.53 Digtime

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DIGTime
```

class DigtimeCls

Digtime commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DigtimeStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower digit-duration limit Range: -100 % to 0 %, Unit: %
- Upper: float: Upper digit-duration limit Range: 0 % to 100 %, Unit: %

get() → DigtimeStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DIGTime
value: DigtimeStruct = driver.configure.afRf.measurement.multiEval.limit.tones.
↪digtime.get()
```

Configures limits for the digit duration in an analyzed tone sequence (DTMF, free dialing and SelCall) .

return
structure: for return value, see the help for DigtimeStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:DIGTime
driver.configure.afRf.measurement.multiEval.limit.tones.digtime.set(enable =␣
↪False, lower = 1.0, upper = 1.0)
```

Configures limits for the digit duration in an analyzed tone sequence (DTMF, free dialing and SelCall) .

param enable
OFF | ON Enables or disables the limit check

param lower
Lower digit-duration limit Range: -100 % to 0 %, Unit: %

param upper
Upper digit-duration limit Range: 0 % to 100 %, Unit: %

6.3.1.2.9.54 Fdeviation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:FDEViation
```

class FdeviationCls

Fdeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FdeviationStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower frequency-deviation limit Range: -100 % to 0 %, Unit: %
- Upper: float: Upper frequency-deviation limit Range: 0 % to 100 %, Unit: %

get() → FdeviationStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:FDEViation
value: FdeviationStruct = driver.configure.afRf.measurement.multiEval.limit.
↪tones.fdeviation.get()
```

Configures limits for the frequency deviation of tones in an analyzed tone sequence (DTMF, free dialing and SelCall) .

return
structure: for return value, see the help for FdeviationStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:FDEViation
driver.configure.afRf.measurement.multiEval.limit.tones.fdeviation.set(enable =␣
↪False, lower = 1.0, upper = 1.0)
```

Configures limits for the frequency deviation of tones in an analyzed tone sequence (DTMF, free dialing and SelCall) .

param enable
OFF | ON Enables or disables the limit check

param lower
Lower frequency-deviation limit Range: -100 % to 0 %, Unit: %

param upper
Upper frequency-deviation limit Range: 0 % to 100 %, Unit: %

6.3.1.2.9.55 Scal

class ScalCls

Scal commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.tones.scal.clone()
```

Subgroups

6.3.1.2.9.56 Fdeviation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:SCAL:FDEViation
```

class FdeviationCls

Fdeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FdeviationStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower frequency-deviation limit Range: -100 % to 0 %, Unit: %
- Upper: float: Upper frequency-deviation limit Range: 0 % to 100 %, Unit: %

get() → FdeviationStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:SCAL:FDEViation
value: FdeviationStruct = driver.configure.afRf.measurement.multiEval.limit.
↪tones.scal.fdeviation.get()
```

Configures limits for the frequency deviation of tones in an analyzed SELCAL sequence.

return
structure: for return value, see the help for FdeviationStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:LIMit:TONes:SCAL:FDEViation
driver.configure.afRf.measurement.multiEval.limit.tones.scal.fdeviation.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Configures limits for the frequency deviation of tones in an analyzed SELCAL sequence.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower frequency-deviation limit Range: -100 % to 0 %, Unit: %

param upper
Upper frequency-deviation limit Range: 0 % to 100 %, Unit: %

6.3.1.2.9.57 Tpause

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:SCAL:TPAuse
```

class TpauseCls

Tpause commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TpauseStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower pause limit Range: 0.1 s to 0.25 s, Unit: s
- Upper: float: Upper pause limit Range: 0.25 s to 3 s, Unit: s

get() → TpauseStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:SCAL:TPAuse
value: TpauseStruct = driver.configure.afRf.measurement.multiEval.limit.tones.
↪scal.tpause.get()
```

Configures limits for the pause between two dual tones of an analyzed SELCAL sequence.

return
structure: for return value, see the help for TpauseStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:SCAL:TPAuse
driver.configure.afRf.measurement.multiEval.limit.tones.scal.tpause.set(enable␣
↪= False, lower = 1.0, upper = 1.0)
```

Configures limits for the pause between two dual tones of an analyzed SELCAL sequence.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower pause limit Range: 0.1 s to 0.25 s, Unit: s

param upper
Upper pause limit Range: 0.25 s to 3 s, Unit: s

6.3.1.2.9.58 Ttime

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:SCAL:TTIMe
```

class TtimeCls

Ttime commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TtimeStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower tone-duration limit Range: 0.1 s to 1 s, Unit: s
- Upper: float: Upper tone-duration limit Range: 1 s to 3 s, Unit: s

get() → TtimeStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:SCAL:TTIMe
value: TtimeStruct = driver.configure.afRf.measurement.multiEval.limit.tones.
↪scal.ttime.get()
```

Configures limits for the tone duration in an analyzed SELCAL sequence.

return
structure: for return value, see the help for TtimeStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:TONes:SCAL:TTIMe
driver.configure.afRf.measurement.multiEval.limit.tones.scal.ttime.set(enable =␣
↪False, lower = 1.0, upper = 1.0)
```

Configures limits for the tone duration in an analyzed SELCAL sequence.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower tone-duration limit Range: 0.1 s to 1 s, Unit: s

param upper
Upper tone-duration limit Range: 1 s to 3 s, Unit: s

6.3.1.2.9.59 Voip

class VoipCls

Voip commands group definition. 6 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.limit.voip.clone()
```

Subgroups

6.3.1.2.9.60 Sinad

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SINad
```

class SinadCls

Sinad commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SinadStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower SINAD limit Range: 0 dB to 140 dB, Unit: dB
- Upper: float: Upper SINAD limit Range: 0 dB to 140 dB, Unit: dB

get() → SinadStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SINad
value: SinadStruct = driver.configure.afRf.measurement.multiEval.limit.voip.
↪sinad.get()
```

Configures limits for the SINAD results, measured via the VoIP input path.

return
structure: for return value, see the help for SinadStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SINad
driver.configure.afRf.measurement.multiEval.limit.voip.sinad.set(enable = False,
↪ lower = 1.0, upper = 1.0)
```

Configures limits for the SINAD results, measured via the VoIP input path.

param enable
OFF | ON Enables or disables the limit check

param lower
Lower SINAD limit Range: 0 dB to 140 dB, Unit: dB

param upper
Upper SINAD limit Range: 0 dB to 140 dB, Unit: dB

6.3.1.2.9.61 SndRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNDRatio
```

class SndRatioCls

SndRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SndRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get() → SndRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNDRatio
value: SndRatioStruct = driver.configure.afRf.measurement.multiEval.limit.voip.
↪sndRatio.get()
```

Configures limits for all SNR results, measured via the VoIP input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SndRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float* = *None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNDRatio
driver.configure.afRf.measurement.multiEval.limit.voip.sndRatio.set(enable =␣
↪False, lower = 1.0, upper = 1.0)
```

Configures limits for all SNR results, measured via the VoIP input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

6.3.1.2.9.62 SnnRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNNRatio
```

class SnnRatioCls

SnnRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SnnRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get() → SnnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNNRatio
value: SnnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.voip.
↪snnRatio.get()
```

Configures limits for all SNR results, measured via the VoIP input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SnnRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float* = *None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNNRatio
driver.configure.afRf.measurement.multiEval.limit.voip.snnRatio.set(enable =␣
↪False, lower = 1.0, upper = 1.0)
```

Configures limits for all SNR results, measured via the VoIP input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

6.3.1.2.9.63 SnRatio

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNRatio
```

class SnRatioCls

SnRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SnRatioStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check.
- Lower: float: Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB
- Upper: float: Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

get() → SnRatioStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNRatio
value: SnRatioStruct = driver.configure.afRf.measurement.multiEval.limit.voip.
↪snRatio.get()
```

Configures limits for all SNR results, measured via the VoIP input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

return
structure: for return value, see the help for SnRatioStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float* = *None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:SNRatio
driver.configure.afRf.measurement.multiEval.limit.voip.snRatio.set(enable =␣
↪False, lower = 1.0, upper = 1.0)
```

Configures limits for all SNR results, measured via the VoIP input path. SNR results include S/N, (S+N) /N and (S+N+D) /N.

param enable
OFF | ON Enables or disables the limit check.

param lower
Lower limit Range: 0.00 dB to 140.00 dB, Unit: dB

param upper
Upper limit Range: 0.00 dB to 140.00 dB, Unit: dB

6.3.1.2.9.64 ThDistortion

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:THDistortion
```

class ThDistortionCls

ThDistortion commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThDistortionStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper THD limit Range: 0.001 % to 100 %, Unit: %

get() → ThDistortionStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:THDistortion
value: ThDistortionStruct = driver.configure.afRf.measurement.multiEval.limit.
↪voip.thDistortion.get()
```

Configures limits for the THD results, measured via the VoIP input path.

return
structure: for return value, see the help for ThDistortionStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:THDistortion
driver.configure.afRf.measurement.multiEval.limit.voip.thDistortion.set(enable␣
↪= False, upper = 1.0)
```

Configures limits for the THD results, measured via the VoIP input path.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper THD limit Range: 0.001 % to 100 %, Unit: %

6.3.1.2.9.65 ThdNoise

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:THDNoise
```

class ThdNoiseCls

ThdNoise commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThdNoiseStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper THD+N limit Range: 0 % to 100 %, Unit: %

get() → ThdNoiseStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:THDNoise
value: ThdNoiseStruct = driver.configure.afRf.measurement.multiEval.limit.voip.
↪thdNoise.get()
```

Configures limits for the THD+N results, measured via the VoIP input path.

return
structure: for return value, see the help for ThdNoiseStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:LIMit:VOIP:THDNoise
driver.configure.afRf.measurement.multiEval.limit.voip.thdNoise.set(enable =␣
↪False, upper = 1.0)
```

Configures limits for the THD+N results, measured via the VoIP input path.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper THD+N limit Range: 0 % to 100 %, Unit: %

6.3.1.2.9.66 Oscilloscope

class OscilloscopeCls

Oscilloscope commands group definition. 8 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.oscilloscope.clone()
```

Subgroups

6.3.1.2.9.67 AudioInput

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:XDIVision
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:MTIMe
```

class AudioInputCls

AudioInput commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_mtime() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:MTIMe
value: float = driver.configure.afRf.measurement.multiEval.oscilloscope.
↪audioInput.get_mtime()
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

return
meas_time: Unit: s

get_xdivision() → Xdivision

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:AIN:XDIVision
value: enums.Xdivision = driver.configure.afRf.measurement.multiEval.
↪oscilloscope.audioInput.get_xdivision()
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

return
xdivision: U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10 | M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

set_mtime(*meas_time: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:MTIMe
driver.configure.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪mtime(meas_time = 1.0)
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

param meas_time
Unit: s

set_xdivision(*xdivision: Xdivision*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:AIN:XDIVision
driver.configure.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪xdivision(xdivision = enums.Xdivision.M1)
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

param xdivision
U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10 | M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

6.3.1.2.9.68 Demodulation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:XDIVision
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:MTIMe
```

class DemodulationCls

Demodulation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_mtime() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MTIMe
value: float = driver.configure.afRf.measurement.multiEval.oscilloscope.
↪demodulation.get_mtime()
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

return
meas_time: Unit: s

get_xdivision() → Xdivision

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:XDIVision
value: enums.Xdivision = driver.configure.afRf.measurement.multiEval.
↪oscilloscope.demodulation.get_xdivision()
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

return
xdivision: U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10 | M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

set_mtime(*meas_time: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MTIMe
driver.configure.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪mtime(meas_time = 1.0)
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

param meas_time
Unit: s

set_xdivision(*xdivision: Xdivision*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:XDIVision
```

(continues on next page)

(continued from previous page)

```
driver.configure.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪xdivision(xdivision = enums.Xdivision.M1)
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

param xdivision
U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10 | M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

6.3.1.2.9.69 Spdif

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:XDIVision
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:MTIMe
```

class SpdifCls

Spdif commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_mtime() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:MTIMe
value: float = driver.configure.afRf.measurement.multiEval.oscilloscope.spdif.
↪get_mtime()
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

return
meas_time: Unit: s

get_xdivision() → Xdivision

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SIN:XDIVision
value: enums.Xdivision = driver.configure.afRf.measurement.multiEval.
↪oscilloscope.spdif.get_xdivision()
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

return
xdivision: U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10 | M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

set_mtime(*meas_time: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:MTIMe
driver.configure.afRf.measurement.multiEval.oscilloscope.spdif.set_mtime(meas_
↪time = 1.0)
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

param meas_time
Unit: s

set_xdivision(*xdivision: Xdivision*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SIN:XDIVision
driver.configure.afRf.measurement.multiEval.oscilloscope.spdif.set_
↪xdivision(xdivision = enums.Xdivision.M1)
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

param xdivision
U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10 | M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

6.3.1.2.9.70 Voip

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:XDIVision
CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:MTIMe
```

class VoipCls

Voip commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_mtime() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:MTIMe
value: float = driver.configure.afRf.measurement.multiEval.oscilloscope.voip.
↪get_mtime()
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

return
meas_time: Unit: s

get_xdivision() → Xdivision

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:XDIVision
value: enums.Xdivision = driver.configure.afRf.measurement.multiEval.
↪oscilloscope.voip.get_xdivision()
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

return
xdivision: U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10

| M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

set_mtime(*meas_time: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:MTIMe
driver.configure.afRf.measurement.multiEval.oscilloscope.voip.set_mtime(meas_
↪time = 1.0)
```

Query the measurement time per input path. The measurement time is the time covered by one complete result trace (10 x-axis divisions) .

param meas_time
Unit: s

set_xdivision(*xdivision: Xdivision*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:XDIVision
driver.configure.afRf.measurement.multiEval.oscilloscope.voip.set_
↪xdivision(xdivision = enums.Xdivision.M1)
```

Configures the x-axis division of the oscilloscope result diagram. The measurement time equals 10 divisions.

param xdivision
U1 | U2 | U5 | U10 | U20 | U50 | U100 | U200 | U500 | M1 | M2 | M5 | M10 | M20 | M50 | M100 | M200 | M500 | S1 Duration of one division. The letters indicate the unit as follows: U = us, M=ms, S=s Example: U20 = 20 us/division

6.3.1.2.9.71 Result

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:OVERview
CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:OSCilloscope
CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:FFT
```

class ResultCls

Result commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_fft() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:FFT
value: bool = driver.configure.afRf.measurement.multiEval.result.get_fft()
```

Enables or disables the measurement of the AF spectrum results.

return
fft_enable: OFF | ON

get_oscilloscope() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:OSCilloscope
value: bool = driver.configure.afRf.measurement.multiEval.result.get_
↪oscilloscope()
```

Enables or disables the measurement of the AF oscilloscope results.

return
osc_enable: OFF | ON

get_overview() → OverviewType

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:OVERview
value: enums.OverviewType = driver.configure.afRf.measurement.multiEval.result.
↪get_overview()
```

Switches between the 'AF Spectrum' and the 'AF Oscilloscope' diagram if 'AF Spectrum' and 'AF Oscilloscope' are enabled results. If no result is enabled, the remote command returns NONE.

return
type_py: NONE | FFT | OSCilloscope

set_fft(*fft_enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:FFT
driver.configure.afRf.measurement.multiEval.result.set_fft(fft_enable = False)
```

Enables or disables the measurement of the AF spectrum results.

param fft_enable
OFF | ON

set_oscilloscope(*osc_enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:OSCilloscope
driver.configure.afRf.measurement.multiEval.result.set_oscilloscope(osc_enable␣
↪= False)
```

Enables or disables the measurement of the AF oscilloscope results.

param osc_enable
OFF | ON

set_overview(*type_py: OverviewType*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RESult:OVERview
driver.configure.afRf.measurement.multiEval.result.set_overview(type_py = enums.
↪OverviewType.FFT)
```

Switches between the 'AF Spectrum' and the 'AF Oscilloscope' diagram if 'AF Spectrum' and 'AF Oscilloscope' are enabled results. If no result is enabled, the remote command returns NONE.

param type_py
NONE | FFT | OSCilloscope

6.3.1.2.9.72 Rf

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:RF:SCOunt
```

class RfCls

Rf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_scount() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RF:SCOunt
value: int = driver.configure.afRf.measurement.multiEval.rf.get_scount()
```

Specifies the number of measurement intervals per measurement cycle for RF results. One measurement interval delivers a single 'Current' value per result.

return
statistic_count: Range: 1 to 1000

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:RF:SCOunt
driver.configure.afRf.measurement.multiEval.rf.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle for RF results. One measurement interval delivers a single 'Current' value per result.

param statistic_count
Range: 1 to 1000

6.3.1.2.9.73 SpdifLeft

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:SINLeft:TMODe
```

class SpdifLeftCls

SpdifLeft commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tmode() → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SINLeft:TMODe
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪spdifLeft.get_tmode()
```

No command help available

return
tone_mode: No help available

set_tmode(*tone_mode: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SINLeft:TMODe
driver.configure.afRf.measurement.multiEval.spdifLeft.set_tmode(tone_mode =␣
↪enums.DigitalToneMode.DCS)
```

No command help available

param tone_mode
No help available

6.3.1.2.9.74 SpdifRight

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:SINRight:TMODe
```

class SpdifRightCls

SpdifRight commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tmode() → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SINRight:TMODe
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪spdifRight.get_tmode()
```

No command help available

return
tone_mode: No help available

set_tmode(*tone_mode: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:SINRight:TMODe
driver.configure.afRf.measurement.multiEval.spdifRight.set_tmode(tone_mode =␣
↪enums.DigitalToneMode.DCS)
```

No command help available

param tone_mode
No help available

6.3.1.2.9.75 Tones

class TonesCls

Tones commands group definition. 38 total commands, 11 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.clone()
```

Subgroups

6.3.1.2.9.76 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.afRf.measurement.multiEval.tones.audioInput.repcap_audioInput_get()
driver.configure.afRf.measurement.multiEval.tones.audioInput.repcap_audioInput_
↪set(repcap.AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.audioInput.clone()
```

Subgroups

6.3.1.2.9.77 Mode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:MODE
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪tones.audioInput.mode.get(audioInput = repcap.AudioInput.Default)
```

Selects a dialing tone mode for an AF input path.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
tone_mode: NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

set(*tone_mode: DigitalToneMode*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:AIN<Nr>:MODE
driver.configure.afRf.measurement.multiEval.tones.audioInput.mode.set(tone_mode␣
↪= enums.DigitalToneMode.DCS, audioInput = repcap.AudioInput.Default)
```

Selects a dialing tone mode for an AF input path.

param tone_mode
NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.1.2.9.78 Dcs

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:ECWord
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:IMODulation
```

class DcsCls

Dcs commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_ec_word() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:ECWord
value: str = driver.configure.afRf.measurement.multiEval.tones.dcs.get_ec_word()
```

Specifies the expected DCS code number.

return
exp_code_word: DCS code number as octal number Not allowed octal numbers are automatically rounded to the closest allowed value, see method RsCma.Source.AfRf.Generator.Tones.Dcs.cword. Range: #Q20 to #Q777

get_imodulation() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:IMODulation
value: bool = driver.configure.afRf.measurement.multiEval.tones.dcs.get_
↪imodulation()
```

Enables or disables the inversion of the FSK demodulation polarity.

return
imodulation: OFF | ON

set_ec_word(*exp_code_word: str*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:ECWord
driver.configure.afRf.measurement.multiEval.tones.dcs.set_ec_word(exp_code_word␣
↪= r1)
```

Specifies the expected DCS code number.

param exp_code_word
DCS code number as octal number Not allowed octal numbers are automatically rounded to the closest allowed value, see method RsCma.Source.AfRf.Generator.Tones.Dcs.cword. Range: #Q20 to #Q777

set_imodulation(*imodulation: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:IMODulation
driver.configure.afRf.measurement.multiEval.tones.dcs.set_
↪imodulation(imodulation = False)
```

Enables or disables the inversion of the FSK demodulation polarity.

param imodulation
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.dcs.clone()
```

Subgroups

6.3.1.2.9.79 Timeout

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:TOUT
```

class TimeoutCls

Timeout commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TimeoutStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the timeout
- Timeout: float: Waiting for a turn-off code is aborted after this time. Range: 0.1 s to 15 s, Unit: s

get() → TimeoutStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:TOUT
value: TimeoutStruct = driver.configure.afRf.measurement.multiEval.tones.dcs.
↪timeout.get()
```

Configures a timeout for completion of the first DCS measurement cycle.

return
structure: for return value, see the help for TimeoutStruct structure arguments.

set(*enable: bool*, *timeout: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DCS:TOUT
driver.configure.afRf.measurement.multiEval.tones.dcs.timeout.set(enable =␣
↪False, timeout = 1.0)
```

Configures a timeout for completion of the first DCS measurement cycle.

param enable
OFF | ON Enables or disables the timeout

param timeout
Waiting for a turn-off code is aborted after this time. Range: 0.1 s to 15 s, Unit: s

6.3.1.2.9.80 Demodulation

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:MODE
```

class DemodulationCls

Demodulation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:MODE
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪tones.demodulation.get_mode()
```

Selects a dialing tone mode for the RF input path.

return
tone_mode: NONE | SELCall | DTMF | FDIA | SCAL | DCS None, SelCall, DTMF, free dialing, SELCAL, DCS

set_mode(*tone_mode: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DEModulation:MODE
driver.configure.afRf.measurement.multiEval.tones.demodulation.set_mode(tone_
↪mode = enums.DigitalToneMode.DCS)
```

Selects a dialing tone mode for the RF input path.

param tone_mode
NONE | SELCall | DTMF | FDIA | SCAL | DCS None, SelCall, DTMF, free dialing, SELCAL, DCS

6.3.1.2.9.81 Dialing

class DialingCls

Dialing commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.dialing.clone()
```

Subgroups

6.3.1.2.9.82 Timeout

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOUT
```

class TimeoutCls

Timeout commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TimeoutStruct

Response structure. Fields:

- Enable: bool: No parameter help available
- Mode: enums.TimeoutMode: No parameter help available
- Timeout: float: No parameter help available

get() → TimeoutStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOUT
value: TimeoutStruct = driver.configure.afRf.measurement.multiEval.tones.
↪dialing.timeout.get()
```

No command help available

return
structure: for return value, see the help for TimeoutStruct structure arguments.

set(*enable: bool*, *mode: TimeoutMode*, *timeout: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOUT
driver.configure.afRf.measurement.multiEval.tones.dialing.timeout.set(enable =␣
↪False, mode = enums.TimeoutMode.AUTO, timeout = 1.0)
```

No command help available

param enable
No help available

param mode
No help available

param timeout
No help available

6.3.1.2.9.83 ToEnd

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOENd
```

class ToEndCls

ToEnd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ToEndStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables the timeout
- Timeout: float: Maximum time interval after the end of a tone and the start of the next tone Range: 0.1 s to 30 s, Unit: s

get() → ToEndStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOENd
value: ToEndStruct = driver.configure.afRf.measurement.multiEval.tones.dialing.
↪toEnd.get()
```

Configures a timeout for waiting for the next tone during a dialing sequence analysis.

return
structure: for return value, see the help for ToEndStruct structure arguments.

set(*enable: bool*, *timeout: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOENd
driver.configure.afRf.measurement.multiEval.tones.dialing.toEnd.set(enable =␣
↪False, timeout = 1.0)
```

Configures a timeout for waiting for the next tone during a dialing sequence analysis.

param enable
OFF | ON Enables the timeout

param timeout
Maximum time interval after the end of a tone and the start of the next tone Range: 0.1 s to 30 s, Unit: s

6.3.1.2.9.84 ToStart

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOSTart
```

class ToStartCls

ToStart commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ToStartStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables the timeout
- Timeout: float: Time interval during which the first tone must be detected Range: 0.8 s to 86400 s, Unit: s

get() → ToStartStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOSTart
value: ToStartStruct = driver.configure.afRf.measurement.multiEval.tones.
↪dialing.toStart.get()
```

Configures a timeout for the detection of the first tone during a dialing sequence analysis.

return
structure: for return value, see the help for ToStartStruct structure arguments.

set(*enable: bool*, *timeout: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DIALing:TOSTart
driver.configure.afRf.measurement.multiEval.tones.dialing.toStart.set(enable =␣
↪False, timeout = 1.0)
```

Configures a timeout for the detection of the first tone during a dialing sequence analysis.

param enable
OFF | ON Enables the timeout

param timeout
Time interval during which the first tone must be detected Range: 0.8 s to 86400 s, Unit: s

6.3.1.2.9.85 Dtmf

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:CFGenerator
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:SLENgth
```

class DtmfCls

Dtmf commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_cfgenerator() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:CFGenerator
value: bool = driver.configure.afRf.measurement.multiEval.tones.dtmf.get_
↪cfgenerator()
```

Couples the DTMF tone settings of the analyzer to the corresponding generator settings.

return
conf_from_gen: OFF | ON

get_slength() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:SLENgth
value: int = driver.configure.afRf.measurement.multiEval.tones.dtmf.get_
↪slength()
```

Specifies the expected length of the analyzed DTMF tone sequence (number of digits) .

return
seq_length: Range: 1 to 42

set_cfgenerator(*conf_from_gen: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:CFGenerator
driver.configure.afRf.measurement.multiEval.tones.dtmf.set_cfgenerator(conf_
↪from_gen = False)
```

Couples the DTMF tone settings of the analyzer to the corresponding generator settings.

param conf_from_gen
OFF | ON

set_slength(*seq_length: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:SLENgth
driver.configure.afRf.measurement.multiEval.tones.dtmf.set_slength(seq_length =␣
↪1)
```

Specifies the expected length of the analyzed DTMF tone sequence (number of digits) .

param seq_length
Range: 1 to 42

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.dtmf.clone()
```

Subgroups

6.3.1.2.9.86 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:FREQuency:RESet
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:FREQuency
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_value() → List[float]

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:FREQuency
value: List[float] = driver.configure.afRf.measurement.multiEval.tones.dtmf.
↪frequency.get_value()
```

Configures the user-defined tone table for DTMF. To enable the table, see method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Dtmf.UserDefined.enable.

return
frequency: Comma-separated list of up to 8 frequencies You can specify fewer than 8 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

reset() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DTMF:FREQuency:RESet
driver.configure.afRf.measurement.multiEval.tones.dtmf.frequency.reset()
```

Triggers a reset of user-defined frequency values to the default frequency values of the DTMF standard.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DTMF:FREQuency:RESet
driver.configure.afRf.measurement.multiEval.tones.dtmf.frequency.reset_with_
↪opc()
```

Triggers a reset of user-defined frequency values to the default frequency values of the DTMF standard.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*frequency: List[float]*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:FREQuency
driver.configure.afRf.measurement.multiEval.tones.dtmf.frequency.set_
↪value(frequency = [1.1, 2.2, 3.3])
```

Configures the user-defined tone table for DTMF. To enable the table, see method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Dtmf.UserDefined.enable.

param frequency
Comma-separated list of up to 8 frequencies You can specify fewer than 8 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

6.3.1.2.9.87 UserDefined

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:DTMF:UDEFined:ENABle
```

class UserDefinedCls

UserDefined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DTMF:UDEFined:ENABle
value: bool = driver.configure.afRf.measurement.multiEval.tones.dtmf.
↪userDefined.get_enable()
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Dtmf.Frequency.value.

return
user_defined: OFF | ON ON: user-defined tone table OFF: default tone table

set_enable(*user_defined: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:DTMF:UDEFined:ENABle
driver.configure.afRf.measurement.multiEval.tones.dtmf.userDefined.set_
↪enable(user_defined = False)
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Dtmf.Frequency.value.

param user_defined
OFF | ON ON: user-defined tone table OFF: default tone table

6.3.1.2.9.88 Fdialing

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:CFGenerator
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:EFRequency
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:TTYPe
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:SLENgth
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:MACCuracy
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:DTLength
```

class FdialingCls

Fdialing commands group definition. 9 total commands, 1 Subgroups, 6 group commands

get_cfgenerator() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:CFGenerator
value: bool = driver.configure.afRf.measurement.multiEval.tones.fdialing.get_
↪cfgenerator()
```

Couples the free-dialing tone settings of the analyzer to the corresponding generator settings.

return
conf_from_gen: OFF | ON

get_dt_length() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:DTLength
value: float = driver.configure.afRf.measurement.multiEval.tones.fdialing.get_
↪dt_length()
```

Sets or queries the minimum length of a single tone that the dialing measurement can detect.

return
dt_length: No help available

get_efrequency() → ExpFrequency

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:EFRequency
value: enums.ExpFrequency = driver.configure.afRf.measurement.multiEval.tones.
↪fdialing.get_efrequency()
```

No command help available

return
exp_frequency: No help available

get_maccuracy() → MeasAccuracy

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:MACCuracy
value: enums.MeasAccuracy = driver.configure.afRf.measurement.multiEval.tones.
↪fdialing.get_maccuracy()
```

Configures the accuracy of the analysis of free-dialing single-tone sequences.

return
meas_accuracy: NORMal | HIGH NORMal: lower tone detection accuracy / minimum pause length HIGH: higher tone detection accuracy / minimum pause length

get_slength() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:SLENgth
value: int = driver.configure.afRf.measurement.multiEval.tones.fdialing.get_
↪slength()
```

Specifies the expected length of the analyzed free-dialing tone sequence (number of digits) .

return
seq_length: Range: 1 to 42

get_ttype() → SingDualToneType

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:TTYPe
value: enums.SingDualToneType = driver.configure.afRf.measurement.multiEval.
↪tones.fdialing.get_ttype()
```

Selects a tone type for free-dialing tone sequence analysis.

return
tone_type: STONe | DTONe Single tones or dual tones

set_cfgenerator(*conf_from_gen: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:CFGenerator
driver.configure.afRf.measurement.multiEval.tones.fdialing.set_cfgenerator(conf_
↪from_gen = False)
```

Couples the free-dialing tone settings of the analyzer to the corresponding generator settings.

param conf_from_gen
OFF | ON

set_dt_length(*dt_length: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:DTLength
driver.configure.afRf.measurement.multiEval.tones.fdialing.set_dt_length(dt_
↪length = 1.0)
```

Sets or queries the minimum length of a single tone that the dialing measurement can detect.

param dt_length
numeric value Range: 0.02 to 0.03 , Unit: s

set_efrequency(*exp_frequency: ExpFrequency*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:EFRequency
driver.configure.afRf.measurement.multiEval.tones.fdialing.set_efrequency(exp_
↪frequency = enums.ExpFrequency.CONF)
```

No command help available

param exp_frequency
No help available

set_maccuracy(*meas_accuracy: MeasAccuracy*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:MACCuracy
driver.configure.afRf.measurement.multiEval.tones.fdialing.set_maccuracy(meas_
↪accuracy = enums.MeasAccuracy.HIGH)
```

Configures the accuracy of the analysis of free-dialing single-tone sequences.

param meas_accuracy
NORMal | HIGH NORMal: lower tone detection accuracy / minimum pause length
HIGH: higher tone detection accuracy / minimum pause length

set_slength(*seq_length: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:SLENgth
driver.configure.afRf.measurement.multiEval.tones.fdialing.set_slength(seq_
↪length = 1)
```

Specifies the expected length of the analyzed free-dialing tone sequence (number of digits) .

param seq_length
Range: 1 to 42

set_ttype(*tone_type: SingDualToneType*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:TTYPe
driver.configure.afRf.measurement.multiEval.tones.fdialing.set_ttype(tone_type␣
↪= enums.SingDualToneType.DTONe)
```

Selects a tone type for free-dialing tone sequence analysis.

param tone_type
STONe | DTONe Single tones or dual tones

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.fdialing.clone()
```

Subgroups

6.3.1.2.9.89 Frequency<FrequencyLobe>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.afRf.measurement.multiEval.tones.fdialing.frequency.repcap_
↪frequencyLobe_get()
driver.configure.afRf.measurement.multiEval.tones.fdialing.frequency.repcap_
↪frequencyLobe_set(repcap.FrequencyLobe.Nr1)
```

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:FREQuency:STONe
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:FREQuency:RANGe
```

class FrequencyCls

Frequency commands group definition. 3 total commands, 1 Subgroups, 2 group commands Repeated Capability: FrequencyLobe, default value after init: FrequencyLobe.Nr1

get_range() → Bandwidth

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:FREQuency:RANGe
value: enums.Bandwidth = driver.configure.afRf.measurement.multiEval.tones.
↪fdialing.frequency.get_range()
```

Sets the frequency range of the lowpass filter and bandpass filter for a free dialing dual-tone measurement as in table Table 'Frequency range and measurement filters'.

return
bandwidth: FR1K | FR1K1 | FR1K2 | FR1K3 | FR1K4 FR1K Frequency range 1.0 kHz FR1K1 | FR1K2 | FR1K3 | FR1K4 Frequency ranges 1.1 kHz to 1.4 kHz

get_stone() → List[float]

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:FREQuency:STONe
value: List[float] = driver.configure.afRf.measurement.multiEval.tones.fdialing.
↪frequency.get_stone()
```

Assigns single-tone frequencies to digits, for analysis of free-dialing tone sequences.

return
tones_frequency: Comma-separated list of 16 frequencies, assigned to the digits 0, 1, ..., 9, A, ..., F Specifying fewer frequencies leaves the remaining digits unchanged. Range: 60 Hz to 4000 Hz, Unit: Hz

set_range(*bandwidth: Bandwidth*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:FREQuency:RANGe
driver.configure.afRf.measurement.multiEval.tones.fdialing.frequency.set_
↪range(bandwidth = enums.Bandwidth.FR1K)
```

Sets the frequency range of the lowpass filter and bandpass filter for a free dialing dual-tone measurement as in table Table 'Frequency range and measurement filters'.

param bandwidth
FR1K | FR1K1 | FR1K2 | FR1K3 | FR1K4 FR1K Frequency range 1.0 kHz FR1K1 | FR1K2 | FR1K3 | FR1K4 Frequency ranges 1.1 kHz to 1.4 kHz

set_stone(*tones_frequency: List[float]*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:FREQuency:STONe
driver.configure.afRf.measurement.multiEval.tones.fdialing.frequency.set_
↪stone(tones_frequency = [1.1, 2.2, 3.3])
```

Assigns single-tone frequencies to digits, for analysis of free-dialing tone sequences.

param tones_frequency
Comma-separated list of 16 frequencies, assigned to the digits 0, 1, ..., 9, A, ..., F Specifying fewer frequencies leaves the remaining digits unchanged. Range: 60 Hz to 4000 Hz, Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.fdialing.frequency.clone()
```

Subgroups

6.3.1.2.9.90 Dtone

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:FDIaling:FREQuency<Nr>:DTONe
```

class DtoneCls

Dtone commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*frequencyLobe=FrequencyLobe.Default*) → List[float]

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:FREQuency<Nr>:DTONe
value: List[float] = driver.configure.afRf.measurement.multiEval.tones.fdialing.
↪frequency.dtone.get(frequencyLobe = repcap.FrequencyLobe.Default)
```

Assigns dual-tone frequencies to digits, for analysis of free-dialing tone sequences.

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
tones_frequency: Comma-separated list of 16 frequencies, assigned to the digits 0, 1, ..., 9, A, ..., F Specifying fewer frequencies leaves the remaining digits unchanged. Range: no=1/2: 60 Hz to 1000 Hz / 1200 Hz to 4000 Hz , Unit: Hz

set(*tones_frequency: List[float]*, *frequencyLobe=FrequencyLobe.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:FDIaling:FREQuency<Nr>:DTONe
driver.configure.afRf.measurement.multiEval.tones.fdialing.frequency.dtone.
↪set(tones_frequency = [1.1, 2.2, 3.3], frequencyLobe = repcap.FrequencyLobe.
↪Default)
```

Assigns dual-tone frequencies to digits, for analysis of free-dialing tone sequences.

param tones_frequency
Comma-separated list of 16 frequencies, assigned to the digits 0, 1, ..., 9, A, ..., F Specifying fewer frequencies leaves the remaining digits unchanged. Range: no=1/2: 60 Hz to 1000 Hz / 1200 Hz to 4000 Hz , Unit: Hz

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

6.3.1.2.9.91 Scal

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:STANdard
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:CFGenerator
```

class ScalCls

Scal commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_cfgenerator() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:CFGenerator
value: bool = driver.configure.afRf.measurement.multiEval.tones.scal.get_
↪cfgenerator()
```

Couples the SELCAL tone settings of the analyzer to the corresponding generator settings.

return
conf_from_gen: OFF | ON

get_standard() → SelCalStandard

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:STANdard
value: enums.SelCalStandard = driver.configure.afRf.measurement.multiEval.tones.
↪scal.get_standard()
```

Selects the SELCAL standard. Selecting the standard also determines the dual-tone frequencies that the CMA expects during a SELCAL measurement.

return
standard: SCAL16 | SCAL32 | UDEFind SCAL16 SELCAL16 standard and corresponding frequencies SCAL32 SELCAL32 standard and corresponding frequencies UDEFinde User-defined dual-tone frequencies

set_cfgenerator(*conf_from_gen: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:CFGenerator
driver.configure.afRf.measurement.multiEval.tones.scal.set_cfgenerator(conf_
↪from_gen = False)
```

Couples the SELCAL tone settings of the analyzer to the corresponding generator settings.

param conf_from_gen
OFF | ON

set_standard(*standard: SelCalStandard*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:STANdard
driver.configure.afRf.measurement.multiEval.tones.scal.set_standard(standard =␣
↪enums.SelCalStandard.SCAL16)
```

Selects the SELCAL standard. Selecting the standard also determines the dual-tone frequencies that the CMA expects during a SELCAL measurement.

param standard
SCAL16 | SCAL32 | UDEFind SCAL16 SELCAL16 standard and corresponding frequencies SCAL32 SELCAL32 standard and corresponding frequencies UDEFinde User-defined dual-tone frequencies

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.scal.clone()
```

Subgroups

6.3.1.2.9.92 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:FREQuency:RESet
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:FREQuency
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_value() → List[float]

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:FREQuency
value: List[float] = driver.configure.afRf.measurement.multiEval.tones.scal.
↪frequency.get_value()
```

Configures the user-defined tone table for SELCAL. To enable the table, see method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Scal.UserDefined.enable.

return
frequency: Comma-separated list of up to 16 frequencies You can specify fewer than 16 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

reset() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SCAL:FREQuency:RESet
driver.configure.afRf.measurement.multiEval.tones.scal.frequency.reset()
```

Triggers a reset of user-defined frequency values to the default frequency values of the SELCAL standard.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SCAL:FREQuency:RESet
driver.configure.afRf.measurement.multiEval.tones.scal.frequency.reset_with_
↪opc()
```

Triggers a reset of user-defined frequency values to the default frequency values of the SELCAL standard.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*frequency: List[float]*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:FREQuency
driver.configure.afRf.measurement.multiEval.tones.scal.frequency.set_
↪value(frequency = [1.1, 2.2, 3.3])
```

Configures the user-defined tone table for SELCAL. To enable the table, see method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Scal.UserDefined.enable.

param frequency
Comma-separated list of up to 16 frequencies You can specify fewer than 16 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

6.3.1.2.9.93 UserDefined

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SCAL:UDEFined:ENABle
```

class UserDefinedCls

UserDefined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SCAL:UDEFined:ENABle
value: bool = driver.configure.afRf.measurement.multiEval.tones.scal.
↪userDefined.get_enable()
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Scal.Frequency.value.

return
user_defined: OFF | ON ON: user-defined tone table OFF: default tone table

set_enable(*user_defined: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SCAL:UDEFined:ENABle
driver.configure.afRf.measurement.multiEval.tones.scal.userDefined.set_
↪enable(user_defined = False)
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.Scal.Frequency.value.

param user_defined
OFF | ON ON: user-defined tone table OFF: default tone table

6.3.1.2.9.94 SelCall

SCPI Command:

```
CONFigure:AFRF:MEASurement<instance>:MEValuation:TONes:SELCall:DTLength
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:CFGenerator
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:SLENgth
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:STANdard
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:MACCuracy
```

class SelCallCls

SelCall commands group definition. 8 total commands, 2 Subgroups, 5 group commands

get_cfgenerator() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SELCall:CFGenerator
value: bool = driver.configure.afRf.measurement.multiEval.tones.selCall.get_
↪cfgenerator()
```

Couples the SelCall tone settings of the analyzer to the corresponding generator settings.

return
conf_from_gen: OFF | ON

get_dt_length() → float

```
# SCPI: CONFigure:AFRF:MEASurement<instance>:MEValuation:TONes:SELCall:DTLength
value: float = driver.configure.afRf.measurement.multiEval.tones.selCall.get_dt_
↪length()
```

Sets the minimum length of a single tone that the dialing measurement can detect.

return
dt_length: No help available

get_maccuracy() → MeasAccuracy

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:MACCuracy
value: enums.MeasAccuracy = driver.configure.afRf.measurement.multiEval.tones.
↪selCall.get_maccuracy()
```

Configures the accuracy of the analysis of SelCall tone sequences.

return
meas_accuracy: NORMal | HIGH NORMal: lower tone detection accuracy / minimum pause length HIGH: higher tone detection accuracy / minimum pause length

get_slength() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:SLENgth
value: int = driver.configure.afRf.measurement.multiEval.tones.selCall.get_
↪slength()
```

Specifies the expected length of the analyzed SelCall tone sequence (number of digits) . By default, the user-defined tone definition is disabled, and the length is fixed (five digits) . If the user-defined tone definition is enabled, you can configure the length.

return
seq_length: Range: 2 to 42

get_standard() → SelCallStandard

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:STANdard
value: enums.SelCallStandard = driver.configure.afRf.measurement.multiEval.
↪tones.selCall.get_standard()
```

Selects a selective-calling standard for SelCall tone sequence analysis.

return
sel_call_standard: CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei

set_cfgenerator(*conf_from_gen: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SELCall:CFGenerator
driver.configure.afRf.measurement.multiEval.tones.selCall.set_cfgenerator(conf_
↪from_gen = False)
```

Couples the SelCall tone settings of the analyzer to the corresponding generator settings.

param conf_from_gen
OFF | ON

set_dt_length(*dt_length: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<instance>:MEValuation:TONes:SELCall:DTLength
driver.configure.afRf.measurement.multiEval.tones.selCall.set_dt_length(dt_
↪length = 1.0)
```

Sets the minimum length of a single tone that the dialing measurement can detect.

param dt_length
numeric value Range: 0.02 to 0.03 , Unit: s

set_maccuracy(*meas_accuracy: MeasAccuracy*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:MACCuracy
driver.configure.afRf.measurement.multiEval.tones.selCall.set_maccuracy(meas_
↪accuracy = enums.MeasAccuracy.HIGH)
```

Configures the accuracy of the analysis of SelCall tone sequences.

param meas_accuracy
NORMal | HIGH NORMal: lower tone detection accuracy / minimum pause length
HIGH: higher tone detection accuracy / minimum pause length

set_slength(*seq_length: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:SLENgth
driver.configure.afRf.measurement.multiEval.tones.selCall.set_slength(seq_
↪length = 1)
```

Specifies the expected length of the analyzed SelCall tone sequence (number of digits) . By default, the user-defined tone definition is disabled, and the length is fixed (five digits) . If the user-defined tone definition is enabled, you can configure the length.

param seq_length
Range: 2 to 42

set_standard(*sel_call_standard: SelCallStandard*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:STANdard
driver.configure.afRf.measurement.multiEval.tones.selCall.set_standard(sel_call_
↪standard = enums.SelCallStandard.CCIR)
```

Selects a selective-calling standard for SelCall tone sequence analysis.

param sel_call_standard
CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.multiEval.tones.selCall.clone()
```

Subgroups

6.3.1.2.9.95 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:FREQuency
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:FREQuency:RESet
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get(*standard: SelCallStandard*) → List[float]

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:FREQuency
value: List[float] = driver.configure.afRf.measurement.multiEval.tones.selCall.
↪frequency.get(standard = enums.SelCallStandard.CCIR)
```

Configures the user-defined tone table for a SelCall standard. To enable the table, see method RsCma.Configure.AfRf. Measurement.MultiEval.Tones.SelCall.UserDefined.enable.

param standard
CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei Selects the SelCall standard

return
frequency: Comma-separated list of up to 16 frequencies, for digit 0 to F You can specify fewer than 16 values to configure only the beginning of the tone table. The *RST values and ranges depend on the SelCall standard and on the digit. The ranges are approximately: default frequency minus 5% to default frequency plus 5%. Unit: Hz

reset() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SELCall:FREQuency:RESet
driver.configure.afRf.measurement.multiEval.tones.selCall.frequency.reset()
```

Triggers a reset of user-defined frequency values to the default frequency values of the selective-calling standard.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SELCall:FREQuency:RESet
driver.configure.afRf.measurement.multiEval.tones.selCall.frequency.reset_with_
↪opc()
```

Triggers a reset of user-defined frequency values to the default frequency values of the selective-calling standard.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set(*standard: SelCallStandard*, *frequency: List[float]*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:FREQuency
driver.configure.afRf.measurement.multiEval.tones.selCall.frequency.
↪set(standard = enums.SelCallStandard.CCIR, frequency = [1.1, 2.2, 3.3])
```

Configures the user-defined tone table for a SelCall standard. To enable the table, see method RsCma.Configure.AfRf. Measurement.MultiEval.Tones.SelCall.UserDefined.enable.

param standard
CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei Selects the SelCall standard

param frequency
Comma-separated list of up to 16 frequencies, for digit 0 to F You can specify fewer than 16 values to configure only the beginning of the tone table. The *RST values and ranges depend on the SelCall standard and on the digit. The ranges are approximately: default frequency minus 5% to default frequency plus 5%. Unit: Hz

6.3.1.2.9.96 UserDefined

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SELCall:UDEFined:ENABle
```

class UserDefinedCls

UserDefined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SELCall:UDEFined:ENABle
value: bool = driver.configure.afRf.measurement.multiEval.tones.selCall.
↪userDefined.get_enable()
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.SelCall.Frequency.set.

return
user_defined: OFF | ON ON: user-defined tone table OFF: default tone table

set_enable(*user_defined: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:MEValuation:TONes:SELCall:UDEFined:ENABle
driver.configure.afRf.measurement.multiEval.tones.selCall.userDefined.set_
↪enable(user_defined = False)
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Configure.AfRf.Measurement. MultiEval.Tones.SelCall.Frequency.set.

param user_defined
OFF | ON ON: user-defined tone table OFF: default tone table

6.3.1.2.9.97 SpdifLeft

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:MODE
```

class SpdifLeftCls

SpdifLeft commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:MODE
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪tones.spdifLeft.get_mode()
```

Selects a dialing tone mode for the left SPDIF channel.

return
tone_mode: NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

set_mode(*tone_mode: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SINLeft:MODE
driver.configure.afRf.measurement.multiEval.tones.spdifLeft.set_mode(tone_mode␣
↪= enums.DigitalToneMode.DCS)
```

Selects a dialing tone mode for the left SPDIF channel.

param tone_mode
NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

6.3.1.2.9.98 SpdifRight

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:MODE
```

class SpdifRightCls

SpdifRight commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:MODE
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪tones.spdifRight.get_mode()
```

Selects a dialing tone mode for the right SPDIF channel.

return
tone_mode: NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

set_mode(*tone_mode: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:SINRight:MODE
driver.configure.afRf.measurement.multiEval.tones.spdifRight.set_mode(tone_mode␣
↪= enums.DigitalToneMode.DCS)
```

Selects a dialing tone mode for the right SPDIF channel.

param tone_mode
NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

6.3.1.2.9.99 Voip

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:MODE
```

class VoipCls

Voip commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → DigitalToneMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:MODE
value: enums.DigitalToneMode = driver.configure.afRf.measurement.multiEval.
↪tones.voip.get_mode()
```

Selects a dialing tone mode for the VoIP path.

return
tone_mode: NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

set_mode(*tone_mode: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:MODE
driver.configure.afRf.measurement.multiEval.tones.voip.set_mode(tone_mode =␣
↪enums.DigitalToneMode.DCS)
```

Selects a dialing tone mode for the VoIP path.

param tone_mode
NONE | SELCall | DTMF | FDIA | SCAL None, SelCall, DTMF, free dialing, SELCAL

6.3.1.2.10 RfCarrier

class RfCarrierCls

RfCarrier commands group definition. 16 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.clone()
```

Subgroups

6.3.1.2.10.1 FreqError

class FreqErrorCls

FreqError commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.freqError.clone()
```

Subgroups

6.3.1.2.10.2 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:MEASured
value: float = driver.configure.afRf.measurement.rfCarrier.freqError.delta.get_
↪measured()
```

Configures the measured reference value.

return
meas_val: Unit: Hz

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.rfCarrier.freqError.
↪delta.get_mode()
```

Configures the reference mode for frequency error.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:USER
value: float = driver.configure.afRf.measurement.rfCarrier.freqError.delta.get_
↪user()
```

Configures the user reference value.

return
user_val: Unit: Hz

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:MODE
driver.configure.afRf.measurement.rfCarrier.freqError.delta.set_mode(mode =␣
↪enums.DeltaMode.MEAS)
```

Configures the reference mode for frequency error.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:USER
driver.configure.afRf.measurement.rfCarrier.freqError.delta.set_user(user_val =␣
↪1.0)
```

Configures the user reference value.

param user_val
Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.freqError.delta.clone()
```

Subgroups

6.3.1.2.10.3 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.freqError.delta.update.set()
```

Triggers the update of the measurement reference value for frequency error.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FERRor:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.freqError.delta.update.set_with_
↪opc()
```

Triggers the update of the measurement reference value for frequency error.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.10.4 Frequency

class FrequencyCls

Frequency commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.frequency.clone()
```

Subgroups

6.3.1.2.10.5 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:MEASured
value: float = driver.configure.afRf.measurement.rfCarrier.frequency.delta.get_
↪measured()
```

Configures the measured reference value.

return
meas_val: Unit: Hz

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.rfCarrier.frequency.
↪delta.get_mode()
```

Configures the reference mode for frequency error.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:USER
value: float = driver.configure.afRf.measurement.rfCarrier.frequency.delta.get_
↪user()
```

Configures the user reference value.

return
user_val: Unit: Hz

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:MODE
driver.configure.afRf.measurement.rfCarrier.frequency.delta.set_mode(mode =␣
↪enums.DeltaMode.MEAS)
```

Configures the reference mode for frequency error.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:USER
driver.configure.afRf.measurement.rfCarrier.frequency.delta.set_user(user_val =␣
↪1.0)
```

Configures the user reference value.

param user_val
Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.frequency.delta.clone()
```

Subgroups

6.3.1.2.10.6 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.frequency.delta.update.set()
```

Triggers the update of the measurement reference value for carrier frequency.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.frequency.delta.update.set_with_
↪opc()
```

Triggers the update of the measurement reference value for carrier frequency.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.10.7 PePower

class PePowerCls

PePower commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.pePower.clone()
```

Subgroups

6.3.1.2.10.8 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:MEASured
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:MODE
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:MEASured
value: float = driver.configure.afRf.measurement.rfCarrier.pePower.delta.get_
↪measured()
```

Configures the measured reference value for power PEP.

return
meas_val: Unit: dBm

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.rfCarrier.pePower.
↪delta.get_mode()
```

Configures the reference mode for carrier power PEP.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:USER
value: float = driver.configure.afRf.measurement.rfCarrier.pePower.delta.get_
↪user()
```

Configures the user reference value for power PEP value.

return
user_val: Unit: dBm

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:MODE
driver.configure.afRf.measurement.rfCarrier.pePower.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

Configures the reference mode for carrier power PEP.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:USER
driver.configure.afRf.measurement.rfCarrier.pePower.delta.set_user(user_val = 1.
↪0)
```

Configures the user reference value for power PEP value.

param user_val
Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.pePower.delta.clone()
```

Subgroups

6.3.1.2.10.9 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.pePower.delta.update.set()
```

Triggers the update of the measurement reference value for carrier power PEP.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:PEPower:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.pePower.delta.update.set_with_opc()
```

Triggers the update of the measurement reference value for carrier power PEP.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.10.10 Power

class PowerCls

Power commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.power.clone()
```

Subgroups

6.3.1.2.10.11 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:MEASured
value: float = driver.configure.afRf.measurement.rfCarrier.power.delta.get_
↪measured()
```

Configures the measured reference value for power root mean square.

return
meas_val: Unit: dBm

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.rfCarrier.power.
↪delta.get_mode()
```

Configures the reference mode for carrier power root mean square.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:USER
value: float = driver.configure.afRf.measurement.rfCarrier.power.delta.get_
↪user()
```

Configures the user reference value for power root mean square.

return
user_val: Unit: dBm

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:MODE
driver.configure.afRf.measurement.rfCarrier.power.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

Configures the reference mode for carrier power root mean square.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:USER
driver.configure.afRf.measurement.rfCarrier.power.delta.set_user(user_val = 1.0)
```

Configures the user reference value for power root mean square.

param user_val
Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfCarrier.power.delta.clone()
```

Subgroups

6.3.1.2.10.12 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.power.delta.update.set()
```

Triggers the update of the measurement reference value for carrier power.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFCarrier:POWer:DELTa:UPDate
driver.configure.afRf.measurement.rfCarrier.power.delta.update.set_with_opc()
```

Triggers the update of the measurement reference value for carrier power.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.11 RfSettings

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFSettings:CONNector
CONFigure:AFRF:MEASurement<Instance>:RFSettings:FREQuency
CONFigure:AFRF:MEASurement<Instance>:RFSettings:ENPower
CONFigure:AFRF:MEASurement<Instance>:RFSettings:EATTenuation
CONFigure:AFRF:MEASurement<Instance>:RFSettings:RFCoupling
CONFigure:AFRF:MEASurement<Instance>:RFSettings:DSPace
CONFigure:AFRF:MEASurement<Instance>:RFSettings:CHANnel
CONFigure:AFRF:MEASurement<Instance>:RFSettings:COFFset
```

class RfSettingsCls

RfSettings commands group definition. 10 total commands, 2 Subgroups, 8 group commands

get_channel() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:CHANnel
value: int = driver.configure.afRf.measurement.rfSettings.get_channel()
```

Specifies the center frequency of the RF analyzer via a channel number, according to the configured channel definition.

return
rf_channel: Range: 0 Ch to 9999 Ch, Unit: Ch

get_coffset() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:COFFset
value: float = driver.configure.afRf.measurement.rfSettings.get_coffset()
```

Shifts the center frequency of the RF analyzer by a channel offset, relative to the frequency defined via the channel number. The range depends on the channel spacing, defined via method RsCma.Configure.AfRf.Measurement.Cdefinition.cspace.

return
channel_offset: Range: -Spacing/2 Hz to +Spacing/2 Hz, Unit: Hz

get_connector() → InputConnector

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:CONNector
value: enums.InputConnector = driver.configure.afRf.measurement.rfSettings.get_
↪connector()
```

Selects the input connector for the measured RF signal.

return
input_connector: RFCom | RFIN

get_dspace() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:DSPace
value: float = driver.configure.afRf.measurement.rfSettings.get_dspace()
```

Configures the duplex spacing between the analyzer frequency and the generator frequency. Frequencyanalyzer = frequencygenerator + duplex spacing This command is only relevant with enabled RF coupling.

return
duplex_space: Range: -500 MHz to 500 MHz, Unit: Hz

get_eattenuation() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:EATTenuation
value: float = driver.configure.afRf.measurement.rfSettings.get_eattenuation()
```

Specifies the external attenuation in the input path. Negative values specify a gain.

return
rf_input_ext_att: Range: -50 dB to 90 dB, Unit: dB

get_envelope_power() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:ENPower
value: float = driver.configure.afRf.measurement.rfSettings.get_envelope_power()
```

Sets the expected nominal power of the measured RF signal. The allowed range depends on several other settings, for example on the selected connector and the external attenuation. For supported ranges, refer to the data sheet.

return
exp_nominal_power: Unit: dBm

get_frequency() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:FREQuency
value: float = driver.configure.afRf.measurement.rfSettings.get_frequency()
```

Sets the center frequency of the RF analyzer.

return
frequency: Range: 100 kHz to 3 GHz, Unit: Hz

get_rf_coupling() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:RFCoupling
value: bool = driver.configure.afRf.measurement.rfSettings.get_rf_coupling()
```

Couples the frequency and channel settings of the analyzer to the corresponding generator settings.

return
rf_coupling: OFF | ON

set_channel(*rf_channel: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:CHANnel
driver.configure.afRf.measurement.rfSettings.set_channel(rf_channel = 1)
```

Specifies the center frequency of the RF analyzer via a channel number, according to the configured channel definition.

param rf_channel
Range: 0 Ch to 9999 Ch, Unit: Ch

set_coffset(*channel_offset: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:COFFset
driver.configure.afRf.measurement.rfSettings.set_coffset(channel_offset = 1.0)
```

Shifts the center frequency of the RF analyzer by a channel offset, relative to the frequency defined via the channel number. The range depends on the channel spacing, defined via method RsCma.Configure.AfRf.Measurement.Cdefinition.cspace.

param channel_offset
Range: -Spacing/2 Hz to +Spacing/2 Hz, Unit: Hz

set_connector(*input_connector: InputConnector*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:CONNector
driver.configure.afRf.measurement.rfSettings.set_connector(input_connector =␣
↪enums.InputConnector.RFCom)
```

Selects the input connector for the measured RF signal.

param input_connector
RFCom | RFIN

set_dspace(*duplex_space: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:DSPace
driver.configure.afRf.measurement.rfSettings.set_dspace(duplex_space = 1.0)
```

Configures the duplex spacing between the analyzer frequency and the generator frequency. Frequencyanalyzer = frequencygenerator + duplex spacing This command is only relevant with enabled RF coupling.

param duplex_space
Range: -500 MHz to 500 MHz, Unit: Hz

set_eattenuation(*rf_input_ext_att: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:EATTenuation
driver.configure.afRf.measurement.rfSettings.set_eattenuation(rf_input_ext_att␣
↪= 1.0)
```

Specifies the external attenuation in the input path. Negative values specify a gain.

param rf_input_ext_att
Range: -50 dB to 90 dB, Unit: dB

set_envelope_power(*exp_nominal_power: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:ENPower
driver.configure.afRf.measurement.rfSettings.set_envelope_power(exp_nominal_
↪power = 1.0)
```

Sets the expected nominal power of the measured RF signal. The allowed range depends on several other settings, for example on the selected connector and the external attenuation. For supported ranges, refer to the data sheet.

param exp_nominal_power
Unit: dBm

set_frequency(*frequency: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:FREQuency
driver.configure.afRf.measurement.rfSettings.set_frequency(frequency = 1.0)
```

Sets the center frequency of the RF analyzer.

param frequency
Range: 100 kHz to 3 GHz, Unit: Hz

set_rf_coupling(*rf_coupling: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:RFCoupling
driver.configure.afRf.measurement.rfSettings.set_rf_coupling(rf_coupling =␣
↪False)
```

Couples the frequency and channel settings of the analyzer to the corresponding generator settings.

param rf_coupling
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.rfSettings.clone()
```

Subgroups

6.3.1.2.11.1 FarFrequency

class FarFrequencyCls

FarFrequency 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.afRf.measurement.rfSettings.farFrequency.clone()
```

Subgroups

6.3.1.2.11.2 Action

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFSettings:FARFrequency:ACTion
```

class ActionCls

Action commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:FARFrequency:ACTion
driver.configure.afRf.measurement.rfSettings.farFrequency.action.set()
```

Sets the reference frequency of the channel definition to the current center frequency of the RF analyzer.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:FARFrequency:ACTion
driver.configure.afRf.measurement.rfSettings.farFrequency.action.set_with_opc()
```

Sets the reference frequency of the channel definition to the current center frequency of the RF analyzer.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.11.3 Rf

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:RFSettings:RF:ENABle
```

class RfCls

Rf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:RF:ENABle
value: bool = driver.configure.afRf.measurement.rfSettings.rf.get_enable()
```

Enables or disables the RF input path.

return
rf_enable: OFF | ON

set_enable(*rf_enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:RFSettings:RF:ENABle
driver.configure.afRf.measurement.rfSettings.rf.set_enable(rf_enable = False)
```

Enables or disables the RF input path.

param rf_enable
OFF | ON

6.3.1.2.12 SearchRoutines

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:MRFLevel
CONFigure:AFRF:MEASurement<Instance>:SROutines:SQValue
CONFigure:AFRF:MEASurement<Instance>:SROutines:SQTYpe
CONFigure:AFRF:MEASurement<Instance>:SROutines:STOLerance
CONFigure:AFRF:MEASurement<Instance>:SROutines:PATH
CONFigure:AFRF:MEASurement<Instance>:SROutines:TOUT
```

class SearchRoutinesCls

SearchRoutines commands group definition. 45 total commands, 9 Subgroups, 6 group commands

get_mrf_level() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:MRFLevel
value: float = driver.configure.afRf.measurement.searchRoutines.get_mrf_level()
```

Configures the maximum RF level for the signal generator. Set a maximum RF level value that matches with the maximum input power of your DUT. A too high value of the RF level can damage your DUT.

return
max_level: Range: -130 dBm to -30 dBm, Unit: dBm

get_path() → SearchRoutinePath

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:PATH
value: enums.SearchRoutinePath = driver.configure.afRf.measurement.
↪searchRoutines.get_path()
```

Configures the path from where the test instrument receives the audio input by the connector or by 'VoIP'.

return
path: AFI1 | AFI2 | VOIP

get_sq_type() → TargetParType

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SQTYpe
value: enums.TargetParType = driver.configure.afRf.measurement.searchRoutines.
↪get_sq_type()
```

Selects the type of audio signal quality to be measured.

return
target_par_type: SINad | SNRatio | SNNRatio | SNDNratio SINad Signal to noise and distortion SNRatio Signal-to-noise ratio S/N SNNRatio (S+N) /N SNDNratio (S+N+D) /N

get_sq_value() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SQValue
value: float = driver.configure.afRf.measurement.searchRoutines.get_sq_value()
```

Configures the target value for the audio signal quality parameters.

return
target_par_val: Range: 1 dB to 46 dB, Unit: dB

get_stolerance() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:STOLerance
value: float = driver.configure.afRf.measurement.searchRoutines.get_stolerance()
```

Configures the maximum allowed deviation of the current signal quality from the average signal quality.

return
tolerance: Unit: dB

get_timeout() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TOUT
value: float = driver.configure.afRf.measurement.searchRoutines.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_mrf_level(*max_level: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:MRFLevel
driver.configure.afRf.measurement.searchRoutines.set_mrf_level(max_level = 1.0)
```

Configures the maximum RF level for the signal generator. Set a maximum RF level value that matches with the maximum input power of your DUT. A too high value of the RF level can damage your DUT.

param max_level
Range: -130 dBm to -30 dBm, Unit: dBm

set_path(*path: SearchRoutinePath*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:PATH
driver.configure.afRf.measurement.searchRoutines.set_path(path = enums.
↪SearchRoutinePath.AFI1)
```

Configures the path from where the test instrument receives the audio input by the connector or by 'VoIP'.

param path
AFI1 | AFI2 | VOIP

set_sq_type(*target_par_type: TargetParType*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SQTYpe
driver.configure.afRf.measurement.searchRoutines.set_sq_type(target_par_type =␣
↪enums.TargetParType.SINad)
```

Selects the type of audio signal quality to be measured.

param target_par_type
SINad | SNRatio | SNNRatio | SNDNratio SINad Signal to noise and distortion SNRatio Signal-to-noise ratio S/N SNNRatio (S+N) /N SNDNratio (S+N+D) /N

set_sq_value(*target_par_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SQValue
driver.configure.afRf.measurement.searchRoutines.set_sq_value(target_par_val =␣
↪1.0)
```

Configures the target value for the audio signal quality parameters.

param target_par_val
Range: 1 dB to 46 dB, Unit: dB

set_stolerance(*tolerance: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:STOLerance
driver.configure.afRf.measurement.searchRoutines.set_stolerance(tolerance = 1.0)
```

Configures the maximum allowed deviation of the current signal quality from the average signal quality.

param tolerance
Unit: dB

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TOUT
driver.configure.afRf.measurement.searchRoutines.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.searchRoutines.clone()
```

Subgroups

6.3.1.2.12.1 Dialing

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:DIALing:ENABle
```

class DialingCls

Dialing commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:DIALing:ENABle
value: bool = driver.configure.afRf.measurement.searchRoutines.dialing.get_
↪enable()
```

Enables the dialing of a tone sequence before the first measurement of the search routine.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:DIALing:ENABle
driver.configure.afRf.measurement.searchRoutines.dialing.set_enable(enable =␣
↪False)
```

Enables the dialing of a tone sequence before the first measurement of the search routine.

param enable
OFF | ON

6.3.1.2.12.2 Limit

class LimitCls

Limit commands group definition. 8 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.searchRoutines.limit.clone()
```

Subgroups

6.3.1.2.12.3 RifBandwidth

class RifBandwidthCls

RifBandwidth commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.searchRoutines.limit.rifBandwidth.clone()
```

Subgroups

6.3.1.2.12.4 BwDisplace

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:RIFBandwidth:BWDisplace
```

class BwDisplaceCls

BwDisplace commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class BwDisplaceStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Lower: float: Range: 1 Hz to 100000 Hz, Unit: Hz
- Upper: float: Range: 1000 Hz to 1 MHz, Unit: Hz

get() → BwDisplaceStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RIFBandwidth:BWDisplace
value: BwDisplaceStruct = driver.configure.afRf.measurement.searchRoutines.
↪limit.rifBandwidth.bwDisplace.get()
```

Enables a limit check and sets limits for the RX bandwidth / RF signal displacement bandwidth.

return
structure: for return value, see the help for BwDisplaceStruct structure arguments.

set(*enable: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RIFBandwidth:BWDisplace
driver.configure.afRf.measurement.searchRoutines.limit.rifBandwidth.bwDisplace.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Enables a limit check and sets limits for the RX bandwidth / RF signal displacement bandwidth.

param enable
OFF | ON

param lower
Range: 1 Hz to 100000 Hz, Unit: Hz

param upper
Range: 1000 Hz to 1 MHz, Unit: Hz

6.3.1.2.12.5 Foffset

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:RIFBandwidth:FOFFset
```

class FoffsetCls

Foffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FoffsetStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Upper: float: Range: 1 Hz to 50000 Hz, Unit: Hz

get() → FoffsetStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RIFBandwidth:FOFFset
value: FoffsetStruct = driver.configure.afRf.measurement.searchRoutines.limit.
↪rifBandwidth.foffset.get()
```

Set the upper limit for the center frequency offset.

return
structure: for return value, see the help for FoffsetStruct structure arguments.

set(*enable: bool*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RIFBandwidth:FOFFset
driver.configure.afRf.measurement.searchRoutines.limit.rifBandwidth.foffset.
↪set(enable = False, upper = 1.0)
```

Set the upper limit for the center frequency offset.

param enable
OFF | ON

param upper
Range: 1 Hz to 50000 Hz, Unit: Hz

6.3.1.2.12.6 Rsensitivity

class RsensitivityCls

Rsensitivity 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.afRf.measurement.searchRoutines.limit.rsensitivity.clone()
```

Subgroups

6.3.1.2.12.7 RsLevel

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:RSENsitivity:RSLevel
```

class RsLevelCls

RsLevel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RsLevelStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Upper limit for the RX sensitivity Range: -120 dBm to -100 dBm, Unit: dBm

get() → RsLevelStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RSENsitivity:RSLevel
value: RsLevelStruct = driver.configure.afRf.measurement.searchRoutines.limit.
↪rsensitivity.rsLevel.get()
```

Configures an upper limit for the measured RX sensitivity.

return
structure: for return value, see the help for RsLevelStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RSENsitivity:RSLevel
driver.configure.afRf.measurement.searchRoutines.limit.rsensitivity.rsLevel.
↪set(enable = False, upper = 1.0)
```

Configures an upper limit for the measured RX sensitivity.

param enable
OFF | ON Enables or disables the limit check

param upper
Upper limit for the RX sensitivity Range: -120 dBm to -100 dBm, Unit: dBm

6.3.1.2.12.8 Rsquelch

class RsquelchCls

Rsquelch commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.searchRoutines.limit.rsquelch.clone()
```

Subgroups

6.3.1.2.12.9 Thysteresis

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:RSQuelch:THYSteresis
```

class ThysteresisCls

Thysteresis commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ThysteresisStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Lower: float: Range: 0.1 dB to 3.0 dB, Unit: dB
- Upper: float: Range: 1.0 dB to 10 dB, Unit: dB

get() → ThysteresisStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RSQuelch:THYSteresis
value: ThysteresisStruct = driver.configure.afRf.measurement.searchRoutines.
↪limit.rsquelch.thysteresis.get()
```

Enables a limit check and sets limits for the squelch hysteresis result.

return
structure: for return value, see the help for ThysteresisStruct structure arguments.

set(*enable: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RSQuelch:THYSteresis
driver.configure.afRf.measurement.searchRoutines.limit.rsquelch.thysteresis.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Enables a limit check and sets limits for the squelch hysteresis result.

param enable
OFF | ON

param lower
Range: 0.1 dB to 3.0 dB, Unit: dB

param upper
Range: 1.0 dB to 10 dB, Unit: dB

6.3.1.2.12.10 Tsensitivity

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:RSQuelch:TSENsitivity
```

class TsensitivityCls

Tsensitivity commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TsensitivityStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Lower: float: Range: -130 dBm to -106 dBm, Unit: dBm
- Upper: float: Range: -108 dBm to -30 dBm, Unit: dBm

get() → TsensitivityStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RSQuelch:TSENsitivity
value: TsensitivityStruct = driver.configure.afRf.measurement.searchRoutines.
↪limit.rsquelch.tsensitivity.get()
```

Enables a limit check and sets limits for the squelch off level.

return
structure: for return value, see the help for TsensitivityStruct structure arguments.

set(*enable: bool*, *lower: float = None*, *upper: float = None*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:RSQuelch:TSENsitivity
driver.configure.afRf.measurement.searchRoutines.limit.rsquelch.tsensitivity.
↪set(enable = False, lower = 1.0, upper = 1.0)
```

Enables a limit check and sets limits for the squelch off level.

param enable
OFF | ON

param lower
Range: -130 dBm to -106 dBm, Unit: dBm

param upper
Range: -108 dBm to -30 dBm, Unit: dBm

6.3.1.2.12.11 Ssnr

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:SSNR
```

class SsnrCls

Ssnr commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SsnrStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Upper: float: Unit: dB
- Lower: float: Unit: dB

get() → SsnrStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:SSNR
value: SsnrStruct = driver.configure.afRf.measurement.searchRoutines.limit.ssnr.
↪get()
```

Enables a limit check and sets limits for the determined SNR.

return
structure: for return value, see the help for SsnrStruct structure arguments.

set(*enable: bool*, *upper: float*, *lower: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:SSNR
driver.configure.afRf.measurement.searchRoutines.limit.ssnr.set(enable = False,␣
↪upper = 1.0, lower = 1.0)
```

Enables a limit check and sets limits for the determined SNR.

param enable
OFF | ON

param upper
Unit: dB

param lower
Unit: dB

6.3.1.2.12.12 Tsensitivity

class TsensitivityCls

Tsensitivity commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.searchRoutines.limit.tsensitivity.clone()
```

Subgroups

6.3.1.2.12.13 AudioOutput

class AudioOutputCls

AudioOutput 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.afRf.measurement.searchRoutines.limit.tsensitivity.audioOutput.
↪clone()
```

Subgroups

6.3.1.2.12.14 Level

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:TSENsitivity:AOUT:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LevelStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Unit: V Upper limit of the level of the generated AF signal for AF/VoIP signal path.

get() → LevelStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:TSENsitivity:AOUT:LEVel
value: LevelStruct = driver.configure.afRf.measurement.searchRoutines.limit.
↪tsensitivity.audioOutput.level.get()
```

Enable and configure the upper limit of the level of the generated AF signal.

return
structure: for return value, see the help for LevelStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:TSENsitivity:AOUT:LEVel
driver.configure.afRf.measurement.searchRoutines.limit.tsensitivity.audioOutput.
↪level.set(enable = False, upper = 1.0)
```

Enable and configure the upper limit of the level of the generated AF signal.

param enable
OFF | ON Enables or disables the limit check

param upper
Unit: V Upper limit of the level of the generated AF signal for AF/VoIP signal path.

6.3.1.2.12.15 Voip

class VoipCls

Voip 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.afRf.measurement.searchRoutines.limit.tsensitivity.voip.clone()
```

Subgroups

6.3.1.2.12.16 Level

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LIMit:TSENsitivity:VOIP:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LevelStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Upper: float: Unit: V Upper limit of the level of the generated AF signal for AF/VoIP signal path.

get() → LevelStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:TSENsitivity:VOIP:LEVel
value: LevelStruct = driver.configure.afRf.measurement.searchRoutines.limit.
↪tsensitivity.voip.level.get()
```

Enable and configure the upper limit of the level of the generated AF signal.

return
structure: for return value, see the help for LevelStruct structure arguments.

set(*enable: bool*, *upper: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>
↪:SROutines:LIMit:TSENsitivity:VOIP:LEVel
driver.configure.afRf.measurement.searchRoutines.limit.tsensitivity.voip.level.
↪set(enable = False, upper = 1.0)
```

Enable and configure the upper limit of the level of the generated AF signal.

param enable
OFF | ON Enables or disables the limit check

param upper
Unit: V Upper limit of the level of the generated AF signal for AF/VoIP signal path.

6.3.1.2.12.17 LrfLevel

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:LRFLevel
```

class LrfLevelCls

LrfLevel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LrfLevelStruct

Response structure. Fields:

- Enabled: bool: OFF | ON
- Min_Level: float: Unit: dBm

get() → LrfLevelStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:LRFLevel
value: LrfLevelStruct = driver.configure.afRf.measurement.searchRoutines.
↪lrfLevel.get()
```

Configures the minimum RF level for the signal generator.

return
structure: for return value, see the help for LrfLevelStruct structure arguments.

set(*enabled: bool*, *min_level: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:LRFLevel
driver.configure.afRf.measurement.searchRoutines.lrfLevel.set(enabled = False,␣
↪min_level = 1.0)
```

Configures the minimum RF level for the signal generator.

param enabled
OFF | ON

param min_level
Unit: dBm

6.3.1.2.12.18 RifBandwidth

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SDMethod
CONFigure:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:RSResults
```

class RifBandwidthCls

RifBandwidth commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_rs_results() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:RSResults
value: bool = driver.configure.afRf.measurement.searchRoutines.rifBandwidth.get_
↪rs_results()
```

If enabled, the result of a previously run RX sensitivity search routine is used. If disabled, the RX sensitivity is determined at the first phase of the search routine.

return
enable: OFF | ON

get_sd_method() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SDMethod
value: bool = driver.configure.afRf.measurement.searchRoutines.rifBandwidth.get_
↪sd_method()
```

If enabled, the search routine follows the TIA-603-D specification determining the signal displacement bandwidth and starting from the RX sensitivity level. If disabled, the search routine uses a noise level method determining the bandwidth, not relying on the RX sensitivity.

return
enable: OFF | ON

set_rs_results(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:RSResults
driver.configure.afRf.measurement.searchRoutines.rifBandwidth.set_rs_
↪results(enable = False)
```

If enabled, the result of a previously run RX sensitivity search routine is used. If disabled, the RX sensitivity is determined at the first phase of the search routine.

param enable
OFF | ON

set_sd_method(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RIFBandwidth:SDMethod
driver.configure.afRf.measurement.searchRoutines.rifBandwidth.set_sd_
↪method(enable = False)
```

If enabled, the search routine follows the TIA-603-D specification determining the signal displacement bandwidth and starting from the RX sensitivity level. If disabled, the search routine uses a noise level method determining the bandwidth, not relying on the RX sensitivity.

param enable
OFF | ON

6.3.1.2.12.19 Rsquelch

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:SOTime
CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:LVLTolerance
CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:EXTent
```

class RsquelchCls

Rsquelch commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_extent() → SearchExtent

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:EXTent
value: enums.SearchExtent = driver.configure.afRf.measurement.searchRoutines.
↪rsquelch.get_extent()
```

Defines the extent of the 'RX Squelch' search routine measurement.

return
search_extent: FULL | OFFLevel | ONLevel FULL Determines the squelch switch-on level and switch-off level. OFFLevel Determines the squelch switch-off level. ONLevel Determines the squelch switch-on level.

get_lvl_tolerance() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:LVLTolerance
value: float = driver.configure.afRf.measurement.searchRoutines.rsquelch.get_
↪lvl_tolerance()
```

Defines the maximum deviation from the measured average audio signal level during the 'Squelch Observation Time' when the squelch is switched off at the DUT.

return
tolerance: Unit: dB

get_so_time() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:SOTime
value: float = driver.configure.afRf.measurement.searchRoutines.rsquelch.get_so_
↪time()
```

Defines the time period for which the audio signal has to be continuously unmuted (or muted) after the DUT has switched off (or on) the squelch. The search routine only returns a positive result for the squelch level if the audio signal quality is detected as continuously high (or low) over that period.

return
sq_observ_time: Range: 1 s to 20 s, Unit: s

set_extent(*search_extent: SearchExtent*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:EXTent
driver.configure.afRf.measurement.searchRoutines.rsquelch.set_extent(search_
↪extent = enums.SearchExtent.FULL)
```

Defines the extent of the 'RX Squelch' search routine measurement.

param search_extent
FULL | OFFLevel | ONLevel FULL Determines the squelch switch-on level and switch-off level. OFFLevel Determines the squelch switch-off level. ONLevel Determines the squelch switch-on level.

set_lvl_tolerance(*tolerance: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:LVLTolerance
driver.configure.afRf.measurement.searchRoutines.rsquelch.set_lvl_
↪tolerance(tolerance = 1.0)
```

Defines the maximum deviation from the measured average audio signal level during the 'Squelch Observation Time' when the squelch is switched off at the DUT.

param tolerance
Unit: dB

set_so_time(*sq_observ_time: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RSQuelch:SOTime
driver.configure.afRf.measurement.searchRoutines.rsquelch.set_so_time(sq_observ_
↪time = 1.0)
```

Defines the time period for which the audio signal has to be continuously unmuted (or muted) after the DUT has switched off (or on) the squelch. The search routine only returns a positive result for the squelch level if the audio signal quality is detected as continuously high (or low) over that period.

param sq_observ_time
Range: 1 s to 20 s, Unit: s

6.3.1.2.12.20 Rx

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:SETime
CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:MODE
```

class RxCls

Rx commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_mode() → SearchRoutine

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:MODE
value: enums.SearchRoutine = driver.configure.afRf.measurement.searchRoutines.
↪rx.get_mode()
```

Selects the RX search routine to be performed.

return
search_routine: RSENsitivity | RSQuelch | RIFBandwidth | SSNR RSENsitivity 'RX Sensitivity' RSQuelch 'RX Squelch' RIFBandwidth 'RX Bandwidth' SSNR 'Switched SNR'

get_se_time() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:SETime
value: float = driver.configure.afRf.measurement.searchRoutines.rx.get_se_time()
```

Waiting time after a change of the signal properties before the measurement is started.

return
setting_time: Unit: s

set_mode(*search_routine: SearchRoutine*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:MODE
driver.configure.afRf.measurement.searchRoutines.rx.set_mode(search_routine =␣
↪enums.SearchRoutine.ADELay)
```

Selects the RX search routine to be performed.

param search_routine
RSENsitivity | RSQuelch | RIFBandwidth | SSNR RSENsitivity 'RX Sensitivity' RSQuelch 'RX Squelch' RIFBandwidth 'RX Bandwidth' SSNR 'Switched SNR'

set_se_time(*setting_time: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:SETime
driver.configure.afRf.measurement.searchRoutines.rx.set_se_time(setting_time =␣
↪1.0)
```

Waiting time after a change of the signal properties before the measurement is started.

param setting_time
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.searchRoutines.rx.clone()
```

Subgroups

6.3.1.2.12.21 AmPoints

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:AMPoints:ENABle
```

class AmPointsCls

AmPoints commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:AMPoints:ENABle
value: bool = driver.configure.afRf.measurement.searchRoutines.rx.amPoints.get_
↪enable()
```

If enabled, the search granularity is decreased (smaller increments/decrements, more measurement steps) .

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:RX:AMPoints:ENABle
driver.configure.afRf.measurement.searchRoutines.rx.amPoints.set_enable(enable␣
↪= False)
```

If enabled, the search granularity is decreased (smaller increments/decrements, more measurement steps) .

param enable
OFF | ON

6.3.1.2.12.22 Ssnr

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:REPetition
CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:SCONdition
CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:MOEXception
CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:SCOunt
CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:AFSource
CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:CREPetition
CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:RCOupling
```

class SsnrCls

Ssnr commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_af_source() → AudioConnector

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:AFSource
value: enums.AudioConnector = driver.configure.afRf.measurement.searchRoutines.
↪ssnr.get_af_source()
```

Specifies the AF source for the switched SNR measurement.

return
af_source: AF1O | AF2O AF1 OUT or AF2 OUT

get_crepetition() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:CREPetition
value: bool = driver.configure.afRf.measurement.searchRoutines.ssnr.get_
↪crepetition()
```

Sets the repetition mode for switched SNR measurements to 'Continuous'.

return
continuous_repetition: OFF | ON

get_mo_exception() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:MOEXception
value: bool = driver.configure.afRf.measurement.searchRoutines.ssnr.get_mo_
↪exception()
```

Specifies if faulty or inaccurate switched SNR measurement results are rejected.

return
meas_on_exception: OFF | ON OFF Faulty results are rejected. ON Results are never rejected.

get_rcoupling() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:RCOupling
value: bool = driver.configure.afRf.measurement.searchRoutines.ssnr.get_
↪rcoupling()
```

Couples the repetition mode of all switched SNR measurements.

return
repetition_coupling: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:REPetition
value: enums.Repeat = driver.configure.afRf.measurement.searchRoutines.ssnr.get_
↪repetition()
```

Selects whether the switched SNR measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous

get_scondition() → StopCondition

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:SCONdition
value: enums.StopCondition = driver.configure.afRf.measurement.searchRoutines.
↪ssnr.get_scondition()
```

Specifies the conditions for an early termination of the measurement.

return
stop_condition: NONE | SLFail

get_scount() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:SCOunt
value: int = driver.configure.afRf.measurement.searchRoutines.ssnr.get_scount()
```

The number of single routine runs used to evaluate the SNR.

return
statistic_count: No help available

set_af_source(*af_source: AudioConnector*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:AFSource
driver.configure.afRf.measurement.searchRoutines.ssnr.set_af_source(af_source =␣
↪enums.AudioConnector.AF1O)
```

Specifies the AF source for the switched SNR measurement.

param af_source
AF1O | AF2O AF1 OUT or AF2 OUT

set_crepetition(*continuous_repetition: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:CREPetition
driver.configure.afRf.measurement.searchRoutines.ssnr.set_
↪crepetition(continuous_repetition = False)
```

Sets the repetition mode for switched SNR measurements to 'Continuous'.

param continuous_repetition
OFF | ON

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:MOEXception
driver.configure.afRf.measurement.searchRoutines.ssnr.set_mo_exception(meas_on_
↪exception = False)
```

Specifies if faulty or inaccurate switched SNR measurement results are rejected.

param meas_on_exception
OFF | ON OFF Faulty results are rejected. ON Results are never rejected.

set_rcoupling(*repetition_coupling: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:RCOupling
driver.configure.afRf.measurement.searchRoutines.ssnr.set_rcoupling(repetition_
↪coupling = False)
```

Couples the repetition mode of all switched SNR measurements.

param repetition_coupling
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:REPetition
driver.configure.afRf.measurement.searchRoutines.ssnr.set_repetition(repetition␣
↪= enums.Repeat.CONTinuous)
```

Selects whether the switched SNR measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous

set_scondition(*stop_condition: StopCondition*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:SCONdition
driver.configure.afRf.measurement.searchRoutines.ssnr.set_scondition(stop_
↪condition = enums.StopCondition.NONE)
```

Specifies the conditions for an early termination of the measurement.

param stop_condition
NONE | SLFail

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:SSNR:SCOunt
driver.configure.afRf.measurement.searchRoutines.ssnr.set_scount(statistic_
↪count = 1)
```

The number of single routine runs used to evaluate the SNR.

param statistic_count
No help available

6.3.1.2.12.23 Tsensitivity

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TRELative
CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TFDeviation
CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TPDeviation
CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TMDepth
CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TTOLerance
CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TPARameter
```

class TsensitivityCls

Tsensitivity commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_tf_deviation() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TFDeviation
value: float = driver.configure.afRf.measurement.searchRoutines.tsensitivity.
↪get_tf_deviation()
```

Specify the target deviation of the modulation signal of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

return
target_freq_dev: Unit: Hz

get_tm_depth() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TMDepth
value: float = driver.configure.afRf.measurement.searchRoutines.tsensitivity.
↪get_tm_depth()
```

Specify the target deviation of the modulation signal of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

return
target_mod_depth: No help available

get_tp_deviation() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TPDeviation
value: float = driver.configure.afRf.measurement.searchRoutines.tsensitivity.
↪get_tp_deviation()
```

Specify the target deviation of the modulation signal of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

return
target_phase_dev: No help available

get_tparameter() → TargetParameter

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TPARameter
value: enums.TargetParameter = driver.configure.afRf.measurement.searchRoutines.
↪tsensitivity.get_tparameter()
```

Configures the target parameter for the deviation value of the demodulated signal.

return
target_parameter: RMS | RMSQ | PPEK | NPEK | PNPA RMS 'RMS' RMSQ 'RMS*Sqrt(2) ' PPEK 'PeakPositive' NPEK 'PeakNegative' PNPA 'PosNegPeakAvg'

get_trelative() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TRELative
value: float = driver.configure.afRf.measurement.searchRoutines.tsensitivity.
↪get_trelative()
```

Configures the relative target value of target deviation depending on the modulation technique.

return
target_relative: Unit: %

get_ttolerance() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TTOLerance
value: float = driver.configure.afRf.measurement.searchRoutines.tsensitivity.
↪get_ttolerance()
```

Configures the maximum allowed deviation of the current target deviation of the demodulated signal.

return
tolerance: Unit: %

set_tf_deviation(*target_freq_dev: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TFDeviation
driver.configure.afRf.measurement.searchRoutines.tsensitivity.set_tf_
↪deviation(target_freq_dev = 1.0)
```

Specify the target deviation of the modulation signal of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

param target_freq_dev
Unit: Hz

set_tm_depth(*target_mod_depth: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TMDepth
driver.configure.afRf.measurement.searchRoutines.tsensitivity.set_tm_
↪depth(target_mod_depth = 1.0)
```

Specify the target deviation of the modulation signal of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

param target_mod_depth
Unit: Hz

set_tp_deviation(*target_phase_dev: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TPDeviation
driver.configure.afRf.measurement.searchRoutines.tsensitivity.set_tp_
↪deviation(target_phase_dev = 1.0)
```

Specify the target deviation of the modulation signal of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

param target_phase_dev
Unit: Hz

set_tparameter(*target_parameter: TargetParameter*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TPARameter
driver.configure.afRf.measurement.searchRoutines.tsensitivity.set_
↪tparameter(target_parameter = enums.TargetParameter.NPEK)
```

Configures the target parameter for the deviation value of the demodulated signal.

param target_parameter
RMS | RMSQ | PPEK | NPEK | PNPA RMS 'RMS' RMSQ 'RMS*Sqrt(2) ' PPEK 'PeakPositive' NPEK 'PeakNegative' PNPA 'PosNegPeakAvg'

set_trelative(*target_relative: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TRELative
driver.configure.afRf.measurement.searchRoutines.tsensitivity.set_
↪trelative(target_relative = 1.0)
```

Configures the relative target value of target deviation depending on the modulation technique.

param target_relative
Unit: %

set_ttolerance(*tolerance: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TSENsitivity:TTOLerance
driver.configure.afRf.measurement.searchRoutines.tsensitivity.set_
↪ttolerance(tolerance = 1.0)
```

Configures the maximum allowed deviation of the current target deviation of the demodulated signal.

param tolerance
Unit: %

6.3.1.2.12.24 Tx

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:MODE
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:SETime
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:MLEVel
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:AFSource
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RFDeviation
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RPDeviation
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RMDepth
```

class TxCls

Tx commands group definition. 8 total commands, 1 Subgroups, 7 group commands

get_af_source() → TxAfSource

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:AFSource
value: enums.TxAfSource = driver.configure.afRf.measurement.searchRoutines.tx.
↪get_af_source()
```

Specifies the signal path, i.e. the AF source, of the AF signal generated in the AFRF signal generator for transmission to the DUT

return
af_source: AF1O | AF2O | VOIP AF1O AF1 OUT AF2O AF2 OUT VOIP VoIP

get_mlevel() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:MLEVel
value: float = driver.configure.afRf.measurement.searchRoutines.tx.get_mlevel()
```

Sets the maximum AF level for the AF/VoIP signal path.

return
max_level: Unit: V

get_mode() → SearchRoutine

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:MODE
value: enums.SearchRoutine = driver.configure.afRf.measurement.searchRoutines.
↪tx.get_mode()
```

Selects the TX search routine to be performed.

return
search_routine: TSENsitivity | SSNR TSENsitivity 'TX Modulation Sensitivity' SSNR 'Switched SNR'

get_rf_deviation() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RFDeviation
value: float = driver.configure.afRf.measurement.searchRoutines.tx.get_rf_
↪deviation()
```

Specify the rated system deviation of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

return
rated_freq_dev: Unit: Hz

get_rm_depth() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RMDepth
value: float = driver.configure.afRf.measurement.searchRoutines.tx.get_rm_
↪depth()
```

Specify the rated system deviation of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

return
rated_mod_depth: No help available

get_rp_deviation() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RPDeviation
value: float = driver.configure.afRf.measurement.searchRoutines.tx.get_rp_
↪deviation()
```

Specify the rated system deviation of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

return
rated_phase_dev: No help available

get_se_time() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:SETime
value: float = driver.configure.afRf.measurement.searchRoutines.tx.get_se_time()
```

Specifies the waiting time after a change of the signal properties before the measurement is started.

return
setting_time: Unit: s

set_af_source(*af_source: TxAfSource*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:AFSource
driver.configure.afRf.measurement.searchRoutines.tx.set_af_source(af_source =␣
↪enums.TxAfSource.AF10)
```

Specifies the signal path, i.e. the AF source, of the AF signal generated in the AFRF signal generator for transmission to the DUT

param af_source
AF1O | AF2O | VOIP AF1O AF1 OUT AF2O AF2 OUT VOIP VoIP

set_mlevel(*max_level: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:MLEVel
driver.configure.afRf.measurement.searchRoutines.tx.set_mlevel(max_level = 1.0)
```

Sets the maximum AF level for the AF/VoIP signal path.

param max_level
Unit: V

set_mode(*search_routine: SearchRoutine*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:MODE
driver.configure.afRf.measurement.searchRoutines.tx.set_mode(search_routine =␣
↪enums.SearchRoutine.ADELay)
```

Selects the TX search routine to be performed.

param search_routine
TSENsitivity | SSNR TSENsitivity 'TX Modulation Sensitivity' SSNR 'Switched SNR'

set_rf_deviation(*rated_freq_dev: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RFDeviation
driver.configure.afRf.measurement.searchRoutines.tx.set_rf_deviation(rated_freq_
↪dev = 1.0)
```

Specify the rated system deviation of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

param rated_freq_dev
Unit: Hz

set_rm_depth(*rated_mod_depth: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RMDepth
driver.configure.afRf.measurement.searchRoutines.tx.set_rm_depth(rated_mod_
↪depth = 1.0)
```

Specify the rated system deviation of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

param rated_mod_depth
Unit: Hz

set_rp_deviation(*rated_phase_dev: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:RPDeviation
driver.configure.afRf.measurement.searchRoutines.tx.set_rp_deviation(rated_
↪phase_dev = 1.0)
```

Specify the rated system deviation of the DUT. Depending on the used modulation technique, the system deviation can be frequency deviation (FM) , modulation depth (AM) and phase deviation (PM) .

param rated_phase_dev
Unit: Hz

set_se_time(*setting_time: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:SETime
driver.configure.afRf.measurement.searchRoutines.tx.set_se_time(setting_time =␣
↪1.0)
```

Specifies the waiting time after a change of the signal properties before the measurement is started.

param setting_time
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.searchRoutines.tx.clone()
```

Subgroups

6.3.1.2.12.25 Demod

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:DEMod:SCHeme
```

class DemodCls

Demod commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_scheme() → Demodulation

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:DEMod:SCHeme
value: enums.Demodulation = driver.configure.afRf.measurement.searchRoutines.tx.
↪demod.get_scheme()
```

Selects the type of demodulation to be performed.

return
scheme: FMSTereo | FM | AM | USB | LSB | PM FMSTereo FM stereo multiplex signal FM, PM, AM Frequency / phase / amplitude modulation USB, LSB Single sideband modulation, upper / lower sideband

set_scheme(*scheme: Demodulation*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SROutines:TX:DEMod:SCHeme
driver.configure.afRf.measurement.searchRoutines.tx.demod.set_scheme(scheme =␣
↪enums.Demodulation.AM)
```

Selects the type of demodulation to be performed.

param scheme
FMSTereo | FM | AM | USB | LSB | PM FMSTereo FM stereo multiplex signal FM, PM,

AM Frequency / phase / amplitude modulation USB, LSB Single sideband modulation, upper / lower sideband

6.3.1.2.13 Sout

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SOUT:SOURce
```

class SoutCls

Sout commands group definition. 3 total commands, 2 Subgroups, 1 group commands

class SourceStruct

Structure for reading output parameters. Fields:

- Source_Left: enums.AudioSource: DEM | DEML Source for the left SPDIF channel DEM Demodulator output (FM, PM, ...) DEML Demodulator output, left channel (FM stereo)
- Source_Right: enums.AudioSource: DEM | DEMR Source for the right SPDIF channel DEM Demodulator output (FM, PM, ...) DEMR Demodulator output, right channel (FM stereo)

get_source() → SourceStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SOUT:SOURce
value: SourceStruct = driver.configure.afRf.measurement.sout.get_source()
```

Sets the audio signal sources for the SPDIF OUT connector.

return
structure: for return value, see the help for SourceStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.sout.clone()
```

Subgroups

6.3.1.2.13.1 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SOUT:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Enable_Left: bool: OFF | ON Switches the left channel off or on
- Enable_Right: bool: OFF | ON Switches the right channel off or on

get() → EnableStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SOUT:ENABle
value: EnableStruct = driver.configure.afRf.measurement.sout.enable.get()
```

Enables or disables the channels of the SPDIF OUT connector.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*enable_left: bool*, *enable_right: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SOUT:ENABle
driver.configure.afRf.measurement.sout.enable.set(enable_left = False, enable_
↪right = False)
```

Enables or disables the channels of the SPDIF OUT connector.

param enable_left
OFF | ON Switches the left channel off or on

param enable_right
OFF | ON Switches the right channel off or on

6.3.1.2.13.2 Level

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SOUT:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LevelStruct

Response structure. Fields:

- Level_Left: float: Level for the left channel Unit: %
- Level_Right: float: Level for the right channel Unit: %

get() → LevelStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SOUT:LEVel
value: LevelStruct = driver.configure.afRf.measurement.sout.level.get()
```

Specifies the output levels for the SPDIF OUT connector.

return
structure: for return value, see the help for LevelStruct structure arguments.

set(*level_left: float*, *level_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SOUT:LEVel
driver.configure.afRf.measurement.sout.level.set(level_left = 1.0, level_right␣
↪= 1.0)
```

Specifies the output levels for the SPDIF OUT connector.

param level_left
Level for the left channel Unit: %

param level_right
Level for the right channel Unit: %

6.3.1.2.14 Spdif

class SpdifCls
Spdif commands group definition. 27 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.clone()
```

Subgroups

6.3.1.2.14.1 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:ENABle
```

class EnableCls
Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct
Response structure. Fields:

- Test_Left: bool: OFF | ON Switches the left channel off or on
- Test_Right: bool: OFF | ON Switches the right channel off or on

get() → EnableStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:ENABle
value: EnableStruct = driver.configure.afRf.measurement.spdif.enable.get()
```

Enables or disables the channels of the SPDIF IN connector.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*test_left: bool*, *test_right: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:ENABle
driver.configure.afRf.measurement.spdif.enable.set(test_left = False, test_
↪right = False)
```

Enables or disables the channels of the SPDIF IN connector.

param test_left
OFF | ON Switches the left channel off or on

param test_right
OFF | ON Switches the right channel off or on

6.3.1.2.14.2 FilterPy

class FilterPyCls

FilterPy commands group definition. 12 total commands, 8 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.filterPy.clone()
```

Subgroups

6.3.1.2.14.3 Bpass

class BpassCls

Bpass commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.filterPy.bpass.clone()
```

Subgroups

6.3.1.2.14.4 Bandwidth

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:BWIDth
```

class BandwidthCls

Bandwidth commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class BandwidthStruct

Response structure. Fields:

- Bandwidth_Left: float: Bandwidth for left SPDIF channel Unit: Hz
- Bandwidth_Right: float: Bandwidth for right SPDIF channel Unit: Hz

get() → BandwidthStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:BWIDth
value: BandwidthStruct = driver.configure.afRf.measurement.spdif.filterPy.bpass.
↪bandwidth.get()
```

Configures the bandwidth of the variable bandpass filter in the SPDIF input path.

return
structure: for return value, see the help for BandwidthStruct structure arguments.

set(*bandwidth_left: float*, *bandwidth_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:BWIDth
driver.configure.afRf.measurement.spdif.filterPy.bpass.bandwidth.set(bandwidth_
↪left = 1.0, bandwidth_right = 1.0)
```

Configures the bandwidth of the variable bandpass filter in the SPDIF input path.

param bandwidth_left
Bandwidth for left SPDIF channel Unit: Hz

param bandwidth_right
Bandwidth for right SPDIF channel Unit: Hz

6.3.1.2.14.5 Cfrequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:CFRequency
```

class CfrequencyCls

Cfrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CfrequencyStruct

Response structure. Fields:

- Frequency_Left: float: Frequency for left SPDIF channel Unit: Hz
- Frequency_Right: float: Frequency for right SPDIF channel Unit: Hz

get() → CfrequencyStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:CFRequency
value: CfrequencyStruct = driver.configure.afRf.measurement.spdif.filterPy.
↪bpass.cfrequency.get()
```

Configures the center frequency of the variable bandpass filter in the SPDIF input path.

return
structure: for return value, see the help for CfrequencyStruct structure arguments.

set(*frequency_left: float*, *frequency_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:CFRequency
driver.configure.afRf.measurement.spdif.filterPy.bpass.cfrequency.set(frequency_
↪left = 1.0, frequency_right = 1.0)
```

Configures the center frequency of the variable bandpass filter in the SPDIF input path.

param frequency_left
Frequency for left SPDIF channel Unit: Hz

param frequency_right
Frequency for right SPDIF channel Unit: Hz

6.3.1.2.14.6 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Enable_Left: bool: OFF | ON Disable or enable filter for left SPDIF channel
- Enable_Right: bool: OFF | ON Disable or enable filter for right SPDIF channel

get() → EnableStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:ENABle
value: EnableStruct = driver.configure.afRf.measurement.spdif.filterPy.bpass.
↪enable.get()
```

Enables or disables the variable bandpass filter in the SPDIF input path.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*enable_left: bool*, *enable_right: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:BPASs:ENABle
driver.configure.afRf.measurement.spdif.filterPy.bpass.enable.set(enable_left =␣
↪False, enable_right = False)
```

Enables or disables the variable bandpass filter in the SPDIF input path.

param enable_left
OFF | ON Disable or enable filter for left SPDIF channel

param enable_right
OFF | ON Disable or enable filter for right SPDIF channel

6.3.1.2.14.7 Dfrequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DFRequency
```

class DfrequencyCls

Dfrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DfrequencyStruct

Response structure. Fields:

- Distor_Freq_Left: float: Frequency for left SPDIF channel Unit: Hz
- Distor_Freq_Right: float: Frequency for right SPDIF channel Unit: Hz

get() → DfrequencyStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DFRequency
value: DfrequencyStruct = driver.configure.afRf.measurement.spdif.filterPy.
↪dfrequency.get()
```

Configures the reference frequency for single-tone measurements via the SPDIF input path.

return
structure: for return value, see the help for DfrequencyStruct structure arguments.

set(*distor_freq_left: float*, *distor_freq_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DFRequency
driver.configure.afRf.measurement.spdif.filterPy.dfrequency.set(distor_freq_
↪left = 1.0, distor_freq_right = 1.0)
```

Configures the reference frequency for single-tone measurements via the SPDIF input path.

param distor_freq_left
Frequency for left SPDIF channel Unit: Hz

param distor_freq_right
Frequency for right SPDIF channel Unit: Hz

6.3.1.2.14.8 Dwidth

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DWIDth
```

class DwidthCls

Dwidth commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class DwidthStruct

Response structure. Fields:

- Dwidth_Left: enums.PwrFilterType: WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth, left channel
- Drelative_Left: enums.Relative: RELative | CONStant Proportional to reference frequency or constant, left channel
- Dwidth_Right: enums.PwrFilterType: WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth, right channel
- Drelative_Right: enums.Relative: RELative | CONStant Proportional to reference frequency or constant, right channel

get() → DwidthStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DWIDth
value: DwidthStruct = driver.configure.afRf.measurement.spdif.filterPy.dwidth.
↪get()
```

Configures the bandwidth of the distortion filter in the SPDIF input path.

return
structure: for return value, see the help for DwidthStruct structure arguments.

set(*dwidth_left: PwrFilterType*, *drelative_left: Relative*, *dwidth_right: PwrFilterType*, *drelative_right: Relative*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DWIDth
driver.configure.afRf.measurement.spdif.filterPy.dwidth.set(dwidth_left = enums.
↪PwrFilterType.NARRow, drelative_left = enums.Relative.CONStant, dwidth_right␣
↪= enums.PwrFilterType.NARRow, drelative_right = enums.Relative.CONStant)
```

Configures the bandwidth of the distortion filter in the SPDIF input path.

param dwidth_left
WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth, left channel

param drelative_left
RELative | CONStant Proportional to reference frequency or constant, left channel

param dwidth_right
WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth, right channel

param drelative_right
RELative | CONStant Proportional to reference frequency or constant, right channel

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.filterPy.dwidth.clone()
```

Subgroups

6.3.1.2.14.9 Sfactor

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DWIDth:SFACtor
```

class SfactorCls

Sfactor commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SfactorStruct

Response structure. Fields:

- Factor_Left: float: Range: 0.001 to 0.005
- Factor_Right: float: Range: 0.001 to 0.005

get() → SfactorStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DWIDth:SFACtor
value: SfactorStruct = driver.configure.afRf.measurement.spdif.filterPy.dwidth.
↪sfactor.get()
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:SIN:FILT:DWID UDEF

return
structure: for return value, see the help for SfactorStruct structure arguments.

set(*factor_left: float*, *factor_right: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:DWIDth:SFACtor
driver.configure.afRf.measurement.spdif.filterPy.dwidth.sfactor.set(factor_left␣
↪= 1.0, factor_right = 1.0)
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:SIN:FILT:DWID UDEF

param factor_left
Range: 0.001 to 0.005

param factor_right
Range: 0.001 to 0.005

6.3.1.2.14.10 Hpass

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:HPASs
```

class HpassCls

Hpass commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class HpassStruct

Response structure. Fields:

- Filter_Left: enums.HighpassFilterExtended: OFF | F6 | F50 | F300 Left SPDIF channel OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz
- Filter_Right: enums.HighpassFilterExtended: OFF | F6 | F50 | F300 Right SPDIF channel

get() → HpassStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:HPASs
value: HpassStruct = driver.configure.afRf.measurement.spdif.filterPy.hpass.
↪get()
```

Configures the highpass filter in the SPDIF input path.

return
structure: for return value, see the help for HpassStruct structure arguments.

set(*filter_left: HighpassFilterExtended*, *filter_right: HighpassFilterExtended*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:HPASs
driver.configure.afRf.measurement.spdif.filterPy.hpass.set(filter_left = enums.
↪HighpassFilterExtended.F300, filter_right = enums.HighpassFilterExtended.F300)
```

Configures the highpass filter in the SPDIF input path.

param filter_left
OFF | F6 | F50 | F300 Left SPDIF channel OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz

param filter_right
OFF | F6 | F50 | F300 Right SPDIF channel

6.3.1.2.14.11 Lpass

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:LPASs
```

class LpassCls

Lpass commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LpassStruct

Response structure. Fields:

- Filter_Left: enums.LowpassFilterExtended: OFF | F255 | F3K | F3K4 | F4K | F15K Left SPDIF channel
- Filter_Right: enums.LowpassFilterExtended: OFF | F255 | F3K | F3K4 | F4K | F15K Right SPDIF channel OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

get() → LpassStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:LPASs
value: LpassStruct = driver.configure.afRf.measurement.spdif.filterPy.lpass.
↪get()
```

Configures the lowpass filter in the SPDIF input path.

return
structure: for return value, see the help for LpassStruct structure arguments.

set(*filter_left: LowpassFilterExtended*, *filter_right: LowpassFilterExtended*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:LPASs
driver.configure.afRf.measurement.spdif.filterPy.lpass.set(filter_left = enums.
↪LowpassFilterExtended.F15K, filter_right = enums.LowpassFilterExtended.F15K)
```

Configures the lowpass filter in the SPDIF input path.

param filter_left
OFF | F255 | F3K | F3K4 | F4K | F15K Left SPDIF channel

param filter_right
OFF | F255 | F3K | F3K4 | F4K | F15K Right SPDIF channel OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

6.3.1.2.14.12 Notch<Notch>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.afRf.measurement.spdif.filterPy.notch.repcap_notch_get()
driver.configure.afRf.measurement.spdif.filterPy.notch.repcap_notch_set(repcap.Notch.Nr1)
```

class NotchCls

Notch commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Notch, default value after init: Notch.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.filterPy.notch.clone()
```

Subgroups

6.3.1.2.14.13 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:NOTCh<Num>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Filter_Left_Enable: bool: OFF | ON
- Filter_Right_Enable: bool: OFF | ON

get(*notch=Notch.Default*) → EnableStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:NOTCh<Num>:ENABle
value: EnableStruct = driver.configure.afRf.measurement.spdif.filterPy.notch.
↪enable.get(notch = repcap.Notch.Default)
```

Enables the notch filters 1, 2 or 3 of the left SPDIF IN or right SPDIF IN connectors.

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*filter_left_enable: bool*, *filter_right_enable: bool*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:NOTCh<Num>:ENABle
driver.configure.afRf.measurement.spdif.filterPy.notch.enable.set(filter_left_
↪enable = False, filter_right_enable = False, notch = repcap.Notch.Default)
```

Enables the notch filters 1, 2 or 3 of the left SPDIF IN or right SPDIF IN connectors.

param filter_left_enable
OFF | ON

param filter_right_enable
OFF | ON

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.14.14 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:NOTCh<Num>:FREQuency
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FrequencyStruct

Response structure. Fields:

- Filter_Left_Frequency: float: Range: 5 Hz to 21000 Hz, Unit: Hz
- Filter_Right_Frequency: float: Range: 5 Hz to 21000 Hz, Unit: Hz

get(*notch=Notch.Default*) → FrequencyStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:NOTCh<Num>:FREQuency
value: FrequencyStruct = driver.configure.afRf.measurement.spdif.filterPy.notch.
↪frequency.get(notch = repcap.Notch.Default)
```

Sets the frequency for the notch filters 1, 2 or 3 of the left SPDIF IN or right SPDIF IN connectors.

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
structure: for return value, see the help for FrequencyStruct structure arguments.

set(*filter_left_frequency: float*, *filter_right_frequency: float*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:NOTCh<Num>:FREQuency
driver.configure.afRf.measurement.spdif.filterPy.notch.frequency.set(filter_
```

(continues on next page)

(continued from previous page)

```
↪left_frequency = 1.0, filter_right_frequency = 1.0, notch = repcap.Notch.
↪Default)
```

Sets the frequency for the notch filters 1, 2 or 3 of the left SPDIF IN or right SPDIF IN connectors.

param filter_left_frequency
Range: 5 Hz to 21000 Hz, Unit: Hz

param filter_right_frequency
Range: 5 Hz to 21000 Hz, Unit: Hz

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.14.15 RobustAuto

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:ROBustauto
```

class RobustAutoCls

RobustAuto commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RobustAutoStruct

Response structure. Fields:

- Automatic_Mode_Left: bool: OFF | ON
- Automatic_Mode_Right: bool: OFF | ON

get() → RobustAutoStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:ROBustauto
value: RobustAutoStruct = driver.configure.afRf.measurement.spdif.filterPy.
↪robustAuto.get()
```

Enables or disables robust automatic mode for distortion signal filtering in the SPDIF input path.

return
structure: for return value, see the help for RobustAutoStruct structure arguments.

set(*automatic_mode_left: bool*, *automatic_mode_right: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:ROBustauto
driver.configure.afRf.measurement.spdif.filterPy.robustAuto.set(automatic_mode_
↪left = False, automatic_mode_right = False)
```

Enables or disables robust automatic mode for distortion signal filtering in the SPDIF input path.

param automatic_mode_left
OFF | ON

param automatic_mode_right
OFF | ON

6.3.1.2.14.16 Weighting

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:WEIGhting
```

class WeightingCls

Weighting commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class WeightingStruct

Response structure. Fields:

- Filter_Left: enums.WeightingFilter: OFF | AWEighting | CCITt | CMESsage Left SPDIF channel OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter
- Filter_Right: enums.WeightingFilter: OFF | AWEighting | CCITt | CMESsage Right SPDIF channel

get() → WeightingStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:WEIGhting
value: WeightingStruct = driver.configure.afRf.measurement.spdif.filterPy.
↪weighting.get()
```

Configures the weighting filter in the SPDIF input path.

return
structure: for return value, see the help for WeightingStruct structure arguments.

set(*filter_left: WeightingFilter*, *filter_right: WeightingFilter*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FILTer:WEIGhting
driver.configure.afRf.measurement.spdif.filterPy.weighting.set(filter_left =␣
↪enums.WeightingFilter.AWEighting, filter_right = enums.WeightingFilter.
↪AWEighting)
```

Configures the weighting filter in the SPDIF input path.

param filter_left
OFF | AWEighting | CCITt | CMESsage Left SPDIF channel OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter

param filter_right
OFF | AWEighting | CCITt | CMESsage Right SPDIF channel

6.3.1.2.14.17 Frequency

class FrequencyCls

Frequency commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.frequency.clone()
```

Subgroups

6.3.1.2.14.18 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 2 Subgroups, 2 group commands

class MeasuredStruct

Structure for reading output parameters. Fields:

- Left_Val: float: Unit: Hz
- Right_Val: float: Unit: Hz

get_measured() → MeasuredStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:MEASured
value: MeasuredStruct = driver.configure.afRf.measurement.spdif.frequency.delta.
↪get_measured()
```

Configures the AF frequency measured reference value for SPDIF path.

return
structure: for return value, see the help for MeasuredStruct structure arguments.

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.spdif.frequency.
↪delta.get_mode()
```

Configures the AF frequency reference mode for SPDIF path.

return
mode: NONE | MEAS | USER

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:MODE
driver.configure.afRf.measurement.spdif.frequency.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

Configures the AF frequency reference mode for SPDIF path.

param mode
NONE | MEAS | USER

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.frequency.delta.clone()
```

Subgroups

6.3.1.2.14.19 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.spdif.frequency.delta.update.set()
```

Triggers the update of the AF frequency measurement reference value for SPDIF path.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.spdif.frequency.delta.update.set_with_opc()
```

Triggers the update of the AF frequency measurement reference value for SPDIF path.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.14.20 User

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:USER
```

class UserCls

User commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class UserStruct

Response structure. Fields:

- Left_Val: float: Unit: Hz
- Right_Val: float: Unit: Hz

get() → UserStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:USER
value: UserStruct = driver.configure.afRf.measurement.spdif.frequency.delta.
↪user.get()
```

Configures the AF frequency user reference value for SPDIF path.

return
structure: for return value, see the help for UserStruct structure arguments.

set(*left_val: float*, *right_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:FREQuency:DELTa:USER
driver.configure.afRf.measurement.spdif.frequency.delta.user.set(left_val = 1.0,
↪ right_val = 1.0)
```

Configures the AF frequency user reference value for SPDIF path.

param left_val
Unit: Hz

param right_val
Unit: Hz

6.3.1.2.14.21 Gcoupling

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:GCOupling
```

class GcouplingCls

Gcoupling commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GcouplingStruct

Response structure. Fields:

- Coupling_Left: enums.GeneratorCoupling: OFF | GEN1 | GEN3 OFF No coupling of left channel GENn Left channel coupled to audio generator n
- Coupling_Right: enums.GeneratorCoupling: OFF | GEN2 | GEN4 OFF No coupling of right channel GENn Right channel coupled to audio generator n

get() → GcouplingStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:GCOupling
value: GcouplingStruct = driver.configure.afRf.measurement.spdif.gcoupling.get()
```

Couples the channels of the SPDIF IN connector to an internal signal generator. The combinations GEN1+GEN4 and GEN3+GEN2 are not allowed.

return
structure: for return value, see the help for GcouplingStruct structure arguments.

set(*coupling_left: GeneratorCoupling*, *coupling_right: GeneratorCoupling*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:GCOupling
driver.configure.afRf.measurement.spdif.gcoupling.set(coupling_left = enums.
↪GeneratorCoupling.GEN1, coupling_right = enums.GeneratorCoupling.GEN1)
```

Couples the channels of the SPDIF IN connector to an internal signal generator. The combinations GEN1+GEN4 and GEN3+GEN2 are not allowed.

param coupling_left
OFF | GEN1 | GEN3 OFF No coupling of left channel GENn Left channel coupled to audio generator n

param coupling_right
OFF | GEN2 | GEN4 OFF No coupling of right channel GENn Right channel coupled to audio generator n

6.3.1.2.14.22 Level

class LevelCls

Level commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.level.clone()
```

Subgroups

6.3.1.2.14.23 Peak

class PeakCls

Peak commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.level.peak.clone()
```

Subgroups

6.3.1.2.14.24 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:MEASured
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:MODE
```

class DeltaCls

Delta commands group definition. 4 total commands, 2 Subgroups, 2 group commands

class MeasuredStruct

Structure for reading output parameters. Fields:

- Left_Meas_Val: float: Unit: %
- Right_Meas_Val: float: Unit: %

get_measured() → MeasuredStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:MEASured
value: MeasuredStruct = driver.configure.afRf.measurement.spdif.level.peak.
↪delta.get_measured()
```

For level peak, configures the AF signal measured reference value of SPDIF path.

return
structure: for return value, see the help for MeasuredStruct structure arguments.

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.spdif.level.peak.
↪delta.get_mode()
```

For level peak, configures the AF signal reference mode of SPDIF path.

return
mode: NONE | MEAS | USER

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:MODE
driver.configure.afRf.measurement.spdif.level.peak.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

For level peak, configures the AF signal reference mode of SPDIF path.

param mode
NONE | MEAS | USER

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.level.peak.delta.clone()
```

Subgroups

6.3.1.2.14.25 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:UPDate
driver.configure.afRf.measurement.spdif.level.peak.delta.update.set()
```

For level peak, triggers the update of the measurement value of SPDIF path.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:UPDate
driver.configure.afRf.measurement.spdif.level.peak.delta.update.set_with_opc()
```

For level peak, triggers the update of the measurement value of SPDIF path.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.14.26 User

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:USER
```

class UserCls

User commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class UserStruct

Response structure. Fields:

- Left_User_Val: float: Unit: %
- Right_User_Val: float: Unit: %

get() → UserStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:USER
value: UserStruct = driver.configure.afRf.measurement.spdif.level.peak.delta.
↪user.get()
```

For level peak, configures the AF signal user reference value of SPDIF path.

return
structure: for return value, see the help for UserStruct structure arguments.

set(*left_user_val: float*, *right_user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:PEAK:DELTa:USER
driver.configure.afRf.measurement.spdif.level.peak.delta.user.set(left_user_val
↪= 1.0, right_user_val = 1.0)
```

For level peak, configures the AF signal user reference value of SPDIF path.

param left_user_val
Unit: %

param right_user_val
Unit: %

6.3.1.2.14.27 Rms

class RmsCls

Rms commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.level.rms.clone()
```

Subgroups

6.3.1.2.14.28 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 2 Subgroups, 2 group commands

class MeasuredStruct

Structure for reading output parameters. Fields:

- Left_Meas_Val: float: Unit: %
- Right_Meas_Val: float: Unit: %

get_measured() → MeasuredStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:MEASured
value: MeasuredStruct = driver.configure.afRf.measurement.spdif.level.rms.delta.
↪get_measured()
```

Configures the AF measured user reference value.

return
structure: for return value, see the help for MeasuredStruct structure arguments.

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.spdif.level.rms.
↪delta.get_mode()
```

Configures the AF level reference mode for SPDIF path.

return
mode: NONE | MEAS | USER

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:MODE
driver.configure.afRf.measurement.spdif.level.rms.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

Configures the AF level reference mode for SPDIF path.

param mode
NONE | MEAS | USER

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.spdif.level.rms.delta.clone()
```

Subgroups

6.3.1.2.14.29 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:UPDate
driver.configure.afRf.measurement.spdif.level.rms.delta.update.set()
```

Triggers the update of the AF level measurement reference value for SPDIF path.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:UPDate
driver.configure.afRf.measurement.spdif.level.rms.delta.update.set_with_opc()
```

Triggers the update of the AF level measurement reference value for SPDIF path.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.14.30 User

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:USER
```

class UserCls

User commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class UserStruct

Response structure. Fields:

- Left_User_Val: float: Unit: %
- Right_User_Val: float: Unit: %

get() → UserStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:USER
value: UserStruct = driver.configure.afRf.measurement.spdif.level.rms.delta.
↪user.get()
```

Configures the AF level user reference value for SPDIF path.

return
structure: for return value, see the help for UserStruct structure arguments.

set(*left_user_val: float*, *right_user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:LEVel:RMS:DELTa:USER
driver.configure.afRf.measurement.spdif.level.rms.delta.user.set(left_user_val␣
↪= 1.0, right_user_val = 1.0)
```

Configures the AF level user reference value for SPDIF path.

param left_user_val
Unit: %

param right_user_val
Unit: %

6.3.1.2.14.31 Tmode

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:SIN:TMODe
```

class TmodeCls

Tmode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TmodeStruct

Response structure. Fields:

- Tone_Mode_Left: enums.ToneMode: No parameter help available
- Tone_Mode_Right: enums.DigitalToneMode: No parameter help available

get() → TmodeStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:TMODe
value: TmodeStruct = driver.configure.afRf.measurement.spdif.tmode.get()
```

No command help available

return
structure: for return value, see the help for TmodeStruct structure arguments.

set(*tone_mode_left: ToneMode*, *tone_mode_right: DigitalToneMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:SIN:TMODe
driver.configure.afRf.measurement.spdif.tmode.set(tone_mode_left = enums.
↪ToneMode.NOISe, tone_mode_right = enums.DigitalToneMode.DCS)
```

No command help available

param tone_mode_left
No help available

param tone_mode_right
No help available

6.3.1.2.15 Voip

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:GCOupling
CONFigure:AFRF:MEASurement<Instance>:VOIP:ENABle
CONFigure:AFRF:MEASurement<Instance>:VOIP:PCODec
CONFigure:AFRF:MEASurement<Instance>:VOIP:FID
```

class VoipCls

Voip commands group definition. 42 total commands, 7 Subgroups, 4 group commands

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:ENABle
value: bool = driver.configure.afRf.measurement.voip.get_enable()
```

Enables or disables the audio signal output of the VoIP input path.

return
enable: OFF | ON

get_fid() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FID
value: float = driver.configure.afRf.measurement.voip.get_fid()
```

Specifies the frequency ID (FID) configured at the DUT.
INTRO_CMD_HELP: Allowed values are, with n = 0 to 39995:

- 0.100 + n * 0.025
- 0.105 + n * 0.025
- 0.110 + n * 0.025
- 0.115 + n * 0.025

Resulting in: 0.100, 0.105, 0.110, 0.115, 0.125, 0.130, 0.135, 0.140, ..., 999.975, 999.980, 999.985, 999.990

return
frequency_id: Frequency ID Not allowed values are rounded to the closest allowed value. Range: 0.1 to 999.99

get_gcoupling() → GeneratorCouplingVoIp

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:GCOupling
value: enums.GeneratorCouplingVoIp = driver.configure.afRf.measurement.voip.get_
↪gcoupling()
```

Couples the audio output of the VoIP input path to an internal signal generator.

return
coupling: OFF | GEN3 | GEN4 OFF No coupling GEN3 Coupled to audio generator 3 GEN4 Coupled to audio generator 4

get_pcodec() → VoIpCodec

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:PCODec
value: enums.VoIpCodec = driver.configure.afRf.measurement.voip.get_pcodec()
```

Queries the type of the pulse code modulation (PCM) codec.

return
pcodec: ALAW | ULAW A-law codec or u-law codec

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:ENABle
driver.configure.afRf.measurement.voip.set_enable(enable = False)
```

Enables or disables the audio signal output of the VoIP input path.

param enable
OFF | ON

set_fid(*frequency_id: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FID
driver.configure.afRf.measurement.voip.set_fid(frequency_id = 1.0)
```

Specifies the frequency ID (FID) configured at the DUT.
INTRO_CMD_HELP: Allowed values are, with n = 0 to 39995:

- 0.100 + n * 0.025
- 0.105 + n * 0.025
- 0.110 + n * 0.025
- 0.115 + n * 0.025

Resulting in: 0.100, 0.105, 0.110, 0.115, 0.125, 0.130, 0.135, 0.140, ..., 999.975, 999.980, 999.985, 999.990

param frequency_id
Frequency ID Not allowed values are rounded to the closest allowed value. Range: 0.1 to 999.99

set_gcoupling(*coupling: GeneratorCouplingVoIp*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:GCOupling
driver.configure.afRf.measurement.voip.set_gcoupling(coupling = enums.
↪GeneratorCouplingVoIp.GEN3)
```

Couples the audio output of the VoIP input path to an internal signal generator.

param coupling
OFF | GEN3 | GEN4 OFF No coupling GEN3 Coupled to audio generator 3 GEN4 Coupled to audio generator 4

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.clone()
```

Subgroups

6.3.1.2.15.1 FilterPy

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:WEIGhting
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DFRequency
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:ROBustauto
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:LPASs
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:HPASs
```

class FilterPyCls

FilterPy commands group definition. 12 total commands, 3 Subgroups, 5 group commands

get_dfrequency() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DFRequency
value: float = driver.configure.afRf.measurement.voip.filterPy.get_dfrequency()
```

Configures the reference frequency for single-tone measurements via the VoIP input path.

return
distor_freq: Range: 1 Hz to 10.5 kHz, Unit: Hz

get_hpass() → HighpassFilterExtended

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:HPASs
value: enums.HighpassFilterExtended = driver.configure.afRf.measurement.voip.
↪filterPy.get_hpass()
```

Configures the highpass filter in the VoIP input path.

return
filter_py: OFF | F6 | F50 | F300 OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz

get_lpass() → LowpassFilterExtended

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:LPASs
value: enums.LowpassFilterExtended = driver.configure.afRf.measurement.voip.
↪filterPy.get_lpass()
```

Configures the lowpass filter in the VoIP input path.

return
filter_py: OFF | F255 | F3K | F3K4 | F4K | F15K OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

get_robust_auto() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:ROBustauto
value: bool = driver.configure.afRf.measurement.voip.filterPy.get_robust_auto()
```

Enables or disables robust automatic mode for distortion signal filtering in the VoIP input path.

return
automatic_mode: OFF | ON

get_weighting() → WeightingFilter

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:WEIGhting
value: enums.WeightingFilter = driver.configure.afRf.measurement.voip.filterPy.
↪get_weighting()
```

Configures the weighting filter in the VoIP input path.

return
filter_py: OFF | AWEighting | CCITt | CMESsage OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter

set_dfrequency(*distor_freq: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DFRequency
driver.configure.afRf.measurement.voip.filterPy.set_dfrequency(distor_freq = 1.
↪0)
```

Configures the reference frequency for single-tone measurements via the VoIP input path.

param distor_freq
Range: 1 Hz to 10.5 kHz, Unit: Hz

set_hpass(*filter_py: HighpassFilterExtended*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:HPASs
driver.configure.afRf.measurement.voip.filterPy.set_hpass(filter_py = enums.
↪HighpassFilterExtended.F300)
```

Configures the highpass filter in the VoIP input path.

param filter_py
OFF | F6 | F50 | F300 OFF Filter disabled F6, F50, F300 Cutoff frequency 6 Hz / 50 Hz / 300 Hz

set_lpass(*filter_py: LowpassFilterExtended*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:LPASs
driver.configure.afRf.measurement.voip.filterPy.set_lpass(filter_py = enums.
↪LowpassFilterExtended.F15K)
```

Configures the lowpass filter in the VoIP input path.

param filter_py
OFF | F255 | F3K | F3K4 | F4K | F15K OFF Filter disabled F255, F3K, F3K4, F4K, F15K Cutoff frequency 255 Hz / 3 kHz / 3.4 kHz / 4 kHz / 15 kHz

set_robust_auto(*automatic_mode: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:ROBustauto
driver.configure.afRf.measurement.voip.filterPy.set_robust_auto(automatic_mode␣
↪= False)
```

Enables or disables robust automatic mode for distortion signal filtering in the VoIP input path.

param automatic_mode
OFF | ON

set_weighting(*filter_py: WeightingFilter*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:WEIGhting
driver.configure.afRf.measurement.voip.filterPy.set_weighting(filter_py = enums.
↪WeightingFilter.AWEighting)
```

Configures the weighting filter in the VoIP input path.

param filter_py
OFF | AWEighting | CCITt | CMESsage OFF Filter disabled AWEighting A-weighting filter CCITt CCITT weighting filter CMESsage C-message weighting filter

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.filterPy.clone()
```

Subgroups

6.3.1.2.15.2 Bpass

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:ENABle
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:CFRequency
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:BWIDth
```

class BpassCls

Bpass commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_bandwidth() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:BWIDth
value: float = driver.configure.afRf.measurement.voip.filterPy.bpass.get_
↪bandwidth()
```

Configures the bandwidth of the variable bandpass filter in the VoIP input path.

return
bandwidth: Range: 20 Hz to 20 kHz, Unit: Hz

get_cfrequency() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:CFRequency
value: float = driver.configure.afRf.measurement.voip.filterPy.bpass.get_
↪cfrequency()
```

Configures the center frequency of the variable bandpass filter in the VoIP input path.

return
frequency: Range: 0 Hz to 21 kHz, Unit: Hz

get_enable() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:ENABle
value: bool = driver.configure.afRf.measurement.voip.filterPy.bpass.get_enable()
```

Enables or disables the variable bandpass filter in the VoIP input path.

return
enable: OFF | ON

set_bandwidth(*bandwidth: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:BWIDth
driver.configure.afRf.measurement.voip.filterPy.bpass.set_bandwidth(bandwidth =␣
↪1.0)
```

Configures the bandwidth of the variable bandpass filter in the VoIP input path.

param bandwidth
Range: 20 Hz to 20 kHz, Unit: Hz

`set_cfrequency`(*frequency: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:CFRequency
driver.configure.afRf.measurement.voip.filterPy.bpass.set_cfrequency(frequency␣
↪= 1.0)
```

Configures the center frequency of the variable bandpass filter in the VoIP input path.

param frequency
Range: 0 Hz to 21 kHz, Unit: Hz

`set_enable`(*enable: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:BPASs:ENABle
driver.configure.afRf.measurement.voip.filterPy.bpass.set_enable(enable = False)
```

Enables or disables the variable bandpass filter in the VoIP input path.

param enable
OFF | ON

6.3.1.2.15.3 Dwidth

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DWIDth
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DWIDth:SFACtor
```

`class DwidthCls`

Dwidth commands group definition. 2 total commands, 0 Subgroups, 2 group commands

`class DwidthStruct`

Response structure. Fields:

- Dwidth: enums.PwrFilterType: WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth
- Relative: enums.Relative: RELative | CONStant Proportional to reference frequency or constant

`get`() → DwidthStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DWIDth
value: DwidthStruct = driver.configure.afRf.measurement.voip.filterPy.dwidth.
↪get()
```

Configures the bandwidth of the distortion filter in the VoIP input path.

return
structure: for return value, see the help for DwidthStruct structure arguments.

`get_sfactor`() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DWIDth:SFACtor
value: float = driver.configure.afRf.measurement.voip.filterPy.dwidth.get_
↪sfactor()
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:VOIP:FILT:DWID UDEF

return
factor: Range: 0.001 to 0.005

set(*dwidth: PwrFilterType*, *relative: Relative*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DWIDth
driver.configure.afRf.measurement.voip.filterPy.dwidth.set(dwidth = enums.
↪PwrFilterType.NARRow, relative = enums.Relative.CONStant)
```

Configures the bandwidth of the distortion filter in the VoIP input path.

param dwidth
WIDE | NARRow | UDEF Wide, narrow or user-defined bandwidth

param relative
RELative | CONStant Proportional to reference frequency or constant

set_sfactor(*factor: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:DWIDth:SFACtor
driver.configure.afRf.measurement.voip.filterPy.dwidth.set_sfactor(factor = 1.0)
```

Sets the distortion filter width factor for a user-defined distortion filter width. CONF:AFRF:MEAS:VOIP:FILT:DWID UDEF

param factor
Range: 0.001 to 0.005

6.3.1.2.15.4 Notch<Notch>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.afRf.measurement.voip.filterPy.notch.repcap_notch_get()
driver.configure.afRf.measurement.voip.filterPy.notch.repcap_notch_set(repcap.Notch.Nr1)
```

class NotchCls

Notch commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Notch, default value after init: Notch.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.filterPy.notch.clone()
```

Subgroups

6.3.1.2.15.5 Enable

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:NOTCh<Num>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*notch=Notch.Default*) → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:NOTCh<Num>:ENABle
value: bool = driver.configure.afRf.measurement.voip.filterPy.notch.enable.
↪get(notch = repcap.Notch.Default)
```

Enables the notch filters 1, 2 or 3 in the VOIP path.

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
filter_enable: OFF | ON

set(*filter_enable: bool*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:NOTCh<Num>:ENABle
driver.configure.afRf.measurement.voip.filterPy.notch.enable.set(filter_enable␣
↪= False, notch = repcap.Notch.Default)
```

Enables the notch filters 1, 2 or 3 in the VOIP path.

param filter_enable
OFF | ON

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.15.6 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:NOTCh<Num>:FREQuency
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*notch=Notch.Default*) → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:NOTCh<Num>:FREQuency
value: float = driver.configure.afRf.measurement.voip.filterPy.notch.frequency.
↪get(notch = repcap.Notch.Default)
```

Sets the frequency for the notch filters 1, 2 or 3 in the VOIP path.

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

return
filter_frequency: Range: 5 Hz to 21000 Hz, Unit: Hz

set(*filter_frequency: float*, *notch=Notch.Default*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FILTer:NOTCh<Num>:FREQuency
driver.configure.afRf.measurement.voip.filterPy.notch.frequency.set(filter_
↪frequency = 1.0, notch = repcap.Notch.Default)
```

Sets the frequency for the notch filters 1, 2 or 3 in the VOIP path.

param filter_frequency
Range: 5 Hz to 21000 Hz, Unit: Hz

param notch
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Notch')

6.3.1.2.15.7 Frequency

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:ATGFrequency
CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency
```

class FrequencyCls

Frequency commands group definition. 6 total commands, 1 Subgroups, 2 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Frequency: float: RF carrier center frequency Unit: Hz
- Channel_Spacing: float: Channel spacing Unit: Hz

get_atg_frequency() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:ATGFrequency
value: bool = driver.configure.afRf.measurement.voip.frequency.get_atg_
↪frequency()
```

Selects whether the carrier center frequency resulting from the FID is copied from the analyzer to the AFRF generator or not.

return
apply_to_gen_rf: OFF | ON

get_value() → ValueStruct

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency
value: ValueStruct = driver.configure.afRf.measurement.voip.frequency.get_
↪value()
```

Queries the RF carrier center frequency and the channel spacing resulting from the configured frequency ID.

return
structure: for return value, see the help for ValueStruct structure arguments.

set_atg_frequency(*apply_to_gen_rf: bool*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:ATGFrequency
driver.configure.afRf.measurement.voip.frequency.set_atg_frequency(apply_to_gen_
↪rf = False)
```

Selects whether the carrier center frequency resulting from the FID is copied from the analyzer to the AFRF generator or not.

param apply_to_gen_rf
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.frequency.clone()
```

Subgroups

6.3.1.2.15.8 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:MEASured
value: float = driver.configure.afRf.measurement.voip.frequency.delta.get_
↪measured()
```

Configures the frequency measured reference value for VoIP.

return
meas_val: Unit: Hz

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.voip.frequency.delta.
↪get_mode()
```

Configures the frequency reference mode for VoIP.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:USER
value: float = driver.configure.afRf.measurement.voip.frequency.delta.get_user()
```

Configures the frequency user reference value for VoIP.

return
user_val: Unit: Hz

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:MODE
driver.configure.afRf.measurement.voip.frequency.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

Configures the frequency reference mode for VoIP.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:USER
driver.configure.afRf.measurement.voip.frequency.delta.set_user(user_val = 1.0)
```

Configures the frequency user reference value for VoIP.

param user_val
Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.frequency.delta.clone()
```

Subgroups

6.3.1.2.15.9 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.voip.frequency.delta.update.set()
```

Triggers the update of the measurement value for VoIP frequency.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:FREQuency:DELTa:UPDate
driver.configure.afRf.measurement.voip.frequency.delta.update.set_with_opc()
```

Triggers the update of the measurement value for VoIP frequency.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.15.10 Level

class LevelCls

Level commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.level.clone()
```

Subgroups

6.3.1.2.15.11 Peak

class PeakCls

Peak commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.level.peak.clone()
```

Subgroups

6.3.1.2.15.12 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:MEASured
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:MODE
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:MEASured
value: float = driver.configure.afRf.measurement.voip.level.peak.delta.get_
↪measured()
```

For level peak, configures the AF signal measured reference value for VoIP.

return
meas_val: Unit: %

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.voip.level.peak.
↪delta.get_mode()
```

For level peak, configures the AF signal reference mode for VoIP.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:USER
value: float = driver.configure.afRf.measurement.voip.level.peak.delta.get_
↪user()
```

For level peak, configures the AF signal user reference value for VoIP.

return
user_val: Unit: %

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:MODE
driver.configure.afRf.measurement.voip.level.peak.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

For level peak, configures the AF signal reference mode for VoIP.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:USER
driver.configure.afRf.measurement.voip.level.peak.delta.set_user(user_val = 1.0)
```

For level peak, configures the AF signal user reference value for VoIP.

param user_val
Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.level.peak.delta.clone()
```

Subgroups

6.3.1.2.15.13 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:UPDate
driver.configure.afRf.measurement.voip.level.peak.delta.update.set()
```

For level peak, triggers the update of the measurement value for VoIP.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:PEAK:DELTa:UPDate
driver.configure.afRf.measurement.voip.level.peak.delta.update.set_with_opc()
```

For level peak, triggers the update of the measurement value for VoIP.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.15.14 Rms

class RmsCls

Rms commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.level.rms.clone()
```

Subgroups

6.3.1.2.15.15 Delta

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:MODE
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:USER
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:MEASured
```

class DeltaCls

Delta commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_measured() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:MEASured
value: float = driver.configure.afRf.measurement.voip.level.rms.delta.get_
↪measured()
```

Configures the AF level measured reference value for VoIP.

return
meas_val: Unit: %

get_mode() → DeltaMode

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:MODE
value: enums.DeltaMode = driver.configure.afRf.measurement.voip.level.rms.delta.
↪get_mode()
```

Configures the AF level reference mode for VoIP.

return
mode: NONE | MEAS | USER

get_user() → float

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:USER
value: float = driver.configure.afRf.measurement.voip.level.rms.delta.get_user()
```

Configures the AF level user reference value for VoIP.

return
user_val: Unit: %

set_mode(*mode: DeltaMode*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:MODE
driver.configure.afRf.measurement.voip.level.rms.delta.set_mode(mode = enums.
↪DeltaMode.MEAS)
```

Configures the AF level reference mode for VoIP.

param mode
NONE | MEAS | USER

set_user(*user_val: float*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:USER
driver.configure.afRf.measurement.voip.level.rms.delta.set_user(user_val = 1.0)
```

Configures the AF level user reference value for VoIP.

param user_val
Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.afRf.measurement.voip.level.rms.delta.clone()
```

Subgroups

6.3.1.2.15.16 Update

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:UPDate
driver.configure.afRf.measurement.voip.level.rms.delta.update.set()
```

Triggers the update of the measurement value for VoIP level.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:LEVel:RMS:DELTa:UPDate
driver.configure.afRf.measurement.voip.level.rms.delta.update.set_with_opc()
```

Triggers the update of the measurement value for VoIP level.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.1.2.15.17 Rssi

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:RSSI:CODE
```

class RssiCls

Rssi commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_code() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:RSSI:CODE
value: int = driver.configure.afRf.measurement.voip.rssi.get_code()
```

Configures the RSSI code.

return
code: No help available

6.3.1.2.15.18 Sip

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:STATe
CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:CODE
CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RESPonse
CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RPRotocol
CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RCAuse
CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RTEXt
```

class SipCls

Sip commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_code() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:CODE
value: int = driver.configure.afRf.measurement.voip.sip.get_code()
```

Queries the code number of the last received SIP response.

return
code: Decimal number, for example 200

get_rcause() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RCAuse
value: str = driver.configure.afRf.measurement.voip.sip.get_rcause()
```

Queries information from the reason head field of the SIP.

return
cause: No help available

get_response() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RESPonse
value: str = driver.configure.afRf.measurement.voip.sip.get_response()
```

Queries the text of the last received SIP response.

return
response: Response string, for example 'OK'

get_rprotocol() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RPRotocol
value: str = driver.configure.afRf.measurement.voip.sip.get_rprotocol()
```

Queries information from the reason head field of the SIP.

return
protocol: No help available

get_rt_ext() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:RTEXt
value: str = driver.configure.afRf.measurement.voip.sip.get_rt_ext()
```

Queries information from the reason head field of the SIP.

return
text: No help available

get_state() → SipState

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:SIP:STATe
value: enums.SipState = driver.configure.afRf.measurement.voip.sip.get_state()
```

Queries the state of the VoIP connection to the DUT.

return
state: TERMinated | ESTablished | ERRor

6.3.1.2.15.19 Squelch

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:SQUelch:STATe
```

class SquelchCls

Squelch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_state() → bool

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:SQUelch:STATe
value: bool = driver.configure.afRf.measurement.voip.squelch.get_state()
```

Queries the receiver squelch state in the VoIP path.

return
state: OFF | ON

6.3.1.2.15.20 Uri

SCPI Command:

```
CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:CMA
CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:IP
CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:PORT
CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:USER
```

class UriCls

Uri commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_cma() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:CMA
value: str = driver.configure.afRf.measurement.voip.uri.get_cma()
```

Specifies the <user> part of the URI of the CMA (‘sip:<user>@<IP address>’) .

return
address: String with user part

get_ip() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:IP
value: str = driver.configure.afRf.measurement.voip.uri.get_ip()
```

Specifies the <IP address> part of the URI of the DUT (‘sip:<user>@<IP address>’) .

return
address: IP address as string

get_port() → int

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:PORT
value: int = driver.configure.afRf.measurement.voip.uri.get_port()
```

Specifies the URI port number of the DUT.

return
port: Range: 1024 to 65.535E+3

get_user() → str

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:USER
value: str = driver.configure.afRf.measurement.voip.uri.get_user()
```

Specifies the <user> part of the URI of the DUT ('sip:<user>@<IP address>') .

return
user: String with user part

set_cma(*address: str*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:CMA
driver.configure.afRf.measurement.voip.uri.set_cma(address = '1')
```

Specifies the <user> part of the URI of the CMA ('sip:<user>@<IP address>') .

param address
String with user part

set_ip(*address: str*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:IP
driver.configure.afRf.measurement.voip.uri.set_ip(address = '1')
```

Specifies the <IP address> part of the URI of the DUT ('sip:<user>@<IP address>') .

param address
IP address as string

set_port(*port: int*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:PORT
driver.configure.afRf.measurement.voip.uri.set_port(port = 1)
```

Specifies the URI port number of the DUT.

param port
Range: 1024 to 65.535E+3

set_user(*user: str*) → None

```
# SCPI: CONFigure:AFRF:MEASurement<Instance>:VOIP:URI:USER
driver.configure.afRf.measurement.voip.uri.set_user(user = '1')
```

Specifies the <user> part of the URI of the DUT ('sip:<user>@<IP address>') .

param user
String with user part

6.3.2 Base

SCPI Command:

```
CONFigure:BASE:SPEaker
CONFigure:BASE:SCENario
```

class BaseCls

Base commands group definition. 28 total commands, 11 Subgroups, 2 group commands

get_scenario() → BaseScenario

```
# SCPI: CONFigure:BASE:SCENario
value: enums.BaseScenario = driver.configure.base.get_scenario()
```

Selects the test scenario. Always select the scenario to be used before configuring and using an application. If you show the display during remote control (for example with the 'Hide Remote Screen' button or SYSTem:DISPlay:UPDate ON) , the execution of this command takes some seconds. Insert a pause into your test script after this command, to ensure that the change has been applied. Or query the setting until the correct new value is returned, before you continue your test script.

return
scenario: TXTest | RXTest | DXTest | SPECtrum | EXPert | AUDio | AVIonics | DTXTest | DRXTest | DSPectrum | DEXPert TXTest | RXTest | DXTest | SPECtrum | EXPert | AUDio | AVIonics Analog scenarios DTXTest | DRXTest | DSPectrum | DEXPert Digital scenarios NOSC Cannot be set, but is returned by a query if no scenario is active.

get_speaker() → bool

```
# SCPI: CONFigure:BASE:SPEaker
value: bool = driver.configure.base.get_speaker()
```

Switches the loudspeaker / headphones on or off.

return
speaker: ON | OFF

set_scenario(*scenario: BaseScenario*) → None

```
# SCPI: CONFigure:BASE:SCENario
driver.configure.base.set_scenario(scenario = enums.BaseScenario.AUDio)
```

Selects the test scenario. Always select the scenario to be used before configuring and using an application. If you show the display during remote control (for example with the 'Hide Remote Screen' button or SYSTem:DISPlay:UPDate ON) , the execution of this command takes some seconds. Insert a pause into your test script after this command, to ensure that the change has been applied. Or query the setting until the correct new value is returned, before you continue your test script.

param scenario
TXTest | RXTest | DXTest | SPECtrum | EXPert | AUDio | AVIonics | DTXTest | DRXTest | DSPectrum | DEXPert TXTest | RXTest | DXTest | SPECtrum | EXPert | AUDio | AVIonics Analog scenarios DTXTest | DRXTest | DSPectrum | DEXPert Digital scenarios NOSC Cannot be set, but is returned by a query if no scenario is active.

set_speaker(*speaker: bool*) → None

```
# SCPI: CONFigure:BASE:SPEaker
driver.configure.base.set_speaker(speaker = False)
```

Switches the loudspeaker / headphones on or off.

param speaker
ON | OFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.base.clone()
```

Subgroups

6.3.2.1 Adjustment

SCPI Command:

```
CONFigure:BASE:ADJustment:TYPE
CONFigure:BASE:ADJustment:VALue
CONFigure:BASE:ADJustment:SAVE
```

class AdjustmentCls

Adjustment commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_type_py() → OscillatorType

```
# SCPI: CONFigure:BASE:ADJustment:TYPE
value: enums.OscillatorType = driver.configure.base.adjustment.get_type_py()
```

Selects the frequency source to be adjusted.

return
adj_type: TCXO | OCXO

get_value() → float

```
# SCPI: CONFigure:BASE:ADJustment:VALue
value: float = driver.configure.base.adjustment.get_value()
```

Adjusts the reference frequency. A lower value decreases the frequency. A higher value increases it.

return
adj_value: Range: 0 to 65535

save() → None

```
# SCPI: CONFigure:BASE:ADJustment:SAVE
driver.configure.base.adjustment.save()
```

Stores the configured adjustment value.

save_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:BASE:ADJustment:SAVE
driver.configure.base.adjustment.save_with_opc()
```

Stores the configured adjustment value.

Same as save, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_type_py(*adj_type: OscillatorType*) → None

```
# SCPI: CONFigure:BASE:ADJustment:TYPE
driver.configure.base.adjustment.set_type_py(adj_type = enums.OscillatorType.
↪OCXO)
```

Selects the frequency source to be adjusted.

param adj_type
TCXO | OCXO

set_value(*adj_value: float*) → None

```
# SCPI: CONFigure:BASE:ADJustment:VALue
driver.configure.base.adjustment.set_value(adj_value = 1.0)
```

Adjusts the reference frequency. A lower value decreases the frequency. A higher value increases it.

param adj_value
Range: 0 to 65535

6.3.2.2 Attenuation

SCPI Command:

```
CONFigure:BASE:ATTenuation:ENABle
CONFigure:BASE:ATTenuation:AWARning
```

class AttenuationCls

Attenuation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_awarning() → bool

```
# SCPI: CONFigure:BASE:ATTenuation:AWARning
value: bool = driver.configure.base.attenuation.get_awarning()
```

Enables or disables an audible warning, to be played if the RF protection circuit is activated.

return
awarning: OFF | ON

get_enable() → bool

```
# SCPI: CONFigure:BASE:ATTenuation:ENABle
value: bool = driver.configure.base.attenuation.get_enable()
```

Enables or disables the internal 17 dB attenuator of the RF COM connector.

return
atten_enable: OFF | ON

set_awarning(*awarning: bool*) → None

```
# SCPI: CONFigure:BASE:ATTenuation:AWARning
driver.configure.base.attenuation.set_awarning(awarning = False)
```

Enables or disables an audible warning, to be played if the RF protection circuit is activated.

param awarning
OFF | ON

set_enable(*atten_enable: bool*) → None

```
# SCPI: CONFigure:BASE:ATTenuation:ENABle
driver.configure.base.attenuation.set_enable(atten_enable = False)
```

Enables or disables the internal 17 dB attenuator of the RF COM connector.

param atten_enable
OFF | ON

6.3.2.3 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.base.audioInput.repcap_audioInput_get()
driver.configure.base.audioInput.repcap_audioInput_set(repcap.AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 4 total commands, 3 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.base.audioInput.clone()
```

Subgroups

6.3.2.3.1 Ecircuitry

SCPI Command:

```
CONFigure:BASE:AIN<nr>:ECIRcuitry
```

class EcircuitryCls

Ecircuitry commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → CircuitryState

```
# SCPI: CONFigure:BASE:AIN<nr>:ECIRcuitry
value: enums.CircuitryState = driver.configure.base.audioInput.ecircuitry.
↪get(audioInput = repcap.AudioInput.Default)
```

Selects the set of AF impedance settings to be used.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
circuitry_state: PASSive | ACTive ACTive: settings for R&S CMA-Z600A (ZBOX commands) PASSive: settings for other equipment (other commands)

set(*circuitry_state: CircuitryState*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:BASE:AIN<nr>:ECIRcuitry
driver.configure.base.audioInput.ecircuitry.set(circuitry_state = enums.
↪CircuitryState.ACTive, audioInput = repcap.AudioInput.Default)
```

Selects the set of AF impedance settings to be used.

param circuitry_state
PASSive | ACTive ACTive: settings for R&S CMA-Z600A (ZBOX commands) PASSive: settings for other equipment (other commands)

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.2.3.2 Limpedance

SCPI Command:

```
CONFigure:BASE:AIN<nr>:LIMPedance
```

class LimpedanceCls

Limpedance commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LimpedanceStruct

Response structure. Fields:

- Enable: bool: OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.
- Impedance: float: Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

get(*audioInput=AudioInput.Default*) → LimpedanceStruct

```
# SCPI: CONFigure:BASE:AIN<nr>:LIMPedance
value: LimpedanceStruct = driver.configure.base.audioInput.limpedance.
↪get(audioInput = repcap.AudioInput.Default)
```

Configures the impedance 'R Load'.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
structure: for return value, see the help for LimpedanceStruct structure arguments.

set(*enable: bool*, *impedance: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:BASE:AIN<nr>:LIMPedance
driver.configure.base.audioInput.limpedance.set(enable = False, impedance = 1.0,
↪ audioInput = repcap.AudioInput.Default)
```

Configures the impedance 'R Load'.

param enable
OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.

param impedance
Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.2.3.3 Zbox

class ZboxCls

Zbox commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.base.audioInput.zbox.clone()
```

Subgroups

6.3.2.3.3.1 Attenuator

SCPI Command:

```
CONFigure:BASE:AIN<nr>:ZBOX:ATTenuator
```

class AttenuatorCls

Attenuator commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → bool

```
# SCPI: CONFigure:BASE:AIN<nr>:ZBOX:ATTenuator
value: bool = driver.configure.base.audioInput.zbox.attenuator.get(audioInput =␣
↪repcap.AudioInput.Default)
```

Specifies whether the AF IN attenuator in the impedance matching unit is on or off.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
enable: OFF | ON Attenuator state

set(*enable: bool*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:BASE:AIN<nr>:ZBOX:ATTenuator
driver.configure.base.audioInput.zbox.attenuator.set(enable = False, audioInput␣
↪= repcap.AudioInput.Default)
```

Specifies whether the AF IN attenuator in the impedance matching unit is on or off.

param enable
OFF | ON Attenuator state

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.2.3.3.2 Impedance

SCPI Command:

```
CONFigure:BASE:AIN<nr>:ZBOX:IMPedance
```

class ImpedanceCls

Impedance commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → Impedance

```
# SCPI: CONFigure:BASE:AIN<nr>:ZBOX:IMPedance
value: enums.Impedance = driver.configure.base.audioInput.zbox.impedance.
↪get(audioInput = repcap.AudioInput.Default)
```

Specifies the impedance that is configured at the impedance matching unit.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
impedance: IHOL | R50 | R150 | R300 | R600 IHOL In high / out low R50 | R150 | R300 | R600 50 Ohm | 150 Ohm | 300 Ohm | 600 Ohm

set(*impedance: Impedance*, *audioInput=AudioInput.Default*) → None

```
# SCPI: CONFigure:BASE:AIN<nr>:ZBOX:IMPedance
driver.configure.base.audioInput.zbox.impedance.set(impedance = enums.Impedance.
↪IHOL, audioInput = repcap.AudioInput.Default)
```

Specifies the impedance that is configured at the impedance matching unit.

param impedance
IHOL | R50 | R150 | R300 | R600 IHOL In high / out low R50 | R150 | R300 | R600 50 Ohm | 150 Ohm | 300 Ohm | 600 Ohm

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.3.2.4 AudioOutput<AudioOutput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.base.audioOutput.repcap_audioOutput_get()
driver.configure.base.audioOutput.repcap_audioOutput_set(repcap.AudioOutput.Nr1)
```

class AudioOutputCls

AudioOutput commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: AudioOutput, default value after init: AudioOutput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.base.audioOutput.clone()
```

Subgroups

6.3.2.4.1 Dimpedance

SCPI Command:

```
CONFigure:BASE:AOUT<nr>:DIMPedance
```

class DimpedanceCls

Dimpedance commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DimpedanceStruct

Response structure. Fields:

- Enable: bool: OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.
- Impedance: float: Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

get(*audioOutput=AudioOutput.Default*) → DimpedanceStruct

```
# SCPI: CONFigure:BASE:AOUT<nr>:DIMPedance
value: DimpedanceStruct = driver.configure.base.audioOutput.dimpedance.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Configures the impedance 'Rin DUT' for passive circuitry.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
structure: for return value, see the help for DimpedanceStruct structure arguments.

set(*enable: bool*, *impedance: float*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:BASE:AOUT<nr>:DIMPedance
driver.configure.base.audioOutput.dimpedance.set(enable = False, impedance = 1.
↪0, audioOutput = repcap.AudioOutput.Default)
```

Configures the impedance 'Rin DUT' for passive circuitry.

param enable
OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.

param impedance
Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.2.4.2 Ecircuitry

SCPI Command:

```
CONFigure:BASE:AOUT<nr>:ECIRcuitry
```

class EcircuitryCls

Ecircuitry commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioOutput=AudioOutput.Default*) → CircuitryState

```
# SCPI: CONFigure:BASE:AOUT<nr>:ECIRcuitry
value: enums.CircuitryState = driver.configure.base.audioOutput.ecircuitry.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Selects the set of AF impedance settings to be used.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
circuitry_state: PASSive | ACTive ACTive: settings for R&S CMA-Z600A (ZBOX commands) PASSive: settings for other equipment (other commands)

set(*circuitry_state: CircuitryState*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:BASE:AOUT<nr>:ECIRcuitry
driver.configure.base.audioOutput.ecircuitry.set(circuitry_state = enums.
↪CircuitryState.ACTive, audioOutput = repcap.AudioOutput.Default)
```

Selects the set of AF impedance settings to be used.

param circuitry_state
PASSive | ACTive ACTive: settings for R&S CMA-Z600A (ZBOX commands) PASSive: settings for other equipment (other commands)

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.2.4.3 Eimpedance

SCPI Command:

```
CONFigure:BASE:AOUT<nr>:EIMPedance
```

class EimpedanceCls

Eimpedance commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EimpedanceStruct

Response structure. Fields:

- Enable: bool: OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.
- Impedance: float: Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

get(*audioOutput=AudioOutput.Default*) → EimpedanceStruct

```
# SCPI: CONFigure:BASE:AOUT<nr>:EIMPedance
value: EimpedanceStruct = driver.configure.base.audioOutput.eimpedance.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Configures the impedance 'R Ext'.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
structure: for return value, see the help for EimpedanceStruct structure arguments.

set(*enable: bool*, *impedance: float*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:BASE:AOUT<nr>:EIMPedance
driver.configure.base.audioOutput.eimpedance.set(enable = False, impedance = 1.
↪0, audioOutput = repcap.AudioOutput.Default)
```

Configures the impedance 'R Ext'.

param enable
OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.

param impedance
Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.2.4.4 Limpedance

SCPI Command:

```
CONFigure:BASE:AOUT<nr>:LIMPedance
```

class LimpedanceCls

Limpedance commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LimpedanceStruct

Response structure. Fields:

- Enable: bool: OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.
- Impedance: float: Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

get(*audioOutput=AudioOutput.Default*) → LimpedanceStruct

```
# SCPI: CONFigure:BASE:AOUT<nr>:LIMPedance
value: LimpedanceStruct = driver.configure.base.audioOutput.limpedance.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Configures the impedance 'R Load'.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
structure: for return value, see the help for LimpedanceStruct structure arguments.

set(*enable: bool*, *impedance: float*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:BASE:AOUT<nr>:LIMPedance
driver.configure.base.audioOutput.limpedance.set(enable = False, impedance = 1.
↪0, audioOutput = repcap.AudioOutput.Default)
```

Configures the impedance 'R Load'.

param enable
OFF | ON ON: Use the configured Impedance. OFF: Ignore the configured Impedance.

param impedance
Range: 1 Ohm to 100E+6 Ohm, Unit: Ohm

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.2.4.5 Zbox

class ZboxCls

Zbox 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.base.audioOutput.zbox.clone()
```

Subgroups

6.3.2.4.5.1 Impedance

SCPI Command:

```
CONFigure:BASE:AOUT<nr>:ZBOX:IMPedance
```

class ImpedanceCls

Impedance commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioOutput=AudioOutput.Default*) → Impedance

```
# SCPI: CONFigure:BASE:AOUT<nr>:ZBOX:IMPedance
value: enums.Impedance = driver.configure.base.audioOutput.zbox.impedance.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Specifies the impedance that is configured at the impedance matching unit.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
impedance: IHOL | R50 | R150 | R300 | R600 IHOL In high / out low R50 | R150 | R300 | R600 50 Ohm | 150 Ohm | 300 Ohm | 600 Ohm

set(*impedance: Impedance*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: CONFigure:BASE:AOUT<nr>:ZBOX:IMPedance
driver.configure.base.audioOutput.zbox.impedance.set(impedance = enums.
↪Impedance.IHOL, audioOutput = repcap.AudioOutput.Default)
```

Specifies the impedance that is configured at the impedance matching unit.

param impedance
IHOL | R50 | R150 | R300 | R600 IHOL In high / out low R50 | R150 | R300 | R600 50 Ohm | 150 Ohm | 300 Ohm | 600 Ohm

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.3.2.5 CmaSound

SCPI Command:

```
CONFigure:BASE:CMASound:VOLume
CONFigure:BASE:CMASound:SOURce
CONFigure:BASE:CMASound:SQUelch
```

class CmaSoundCls

CmaSound commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_source() → SoundSource

```
# SCPI: CONFigure:BASE:CMASound:SOURce
value: enums.SoundSource = driver.configure.base.cmaSound.get_source()
```

Selects the audio source to be connected to the loudspeaker / headphones.

return
sound_source: GENone | GENThree | AFONe | SPDif | DEModulator | LAN | AVIO GENone Generator 1 + generator 2 GENThree Generator 3 + generator 4 AFONe AF1 IN + AF2 IN SPDif SPDIF IN L + R DEModulator Demodulator output LAN LAN connector (voice over IP) AVIO Avionic generator

get_squelch() → bool

```
# SCPI: CONFigure:BASE:CMASound:SQUelch
value: bool = driver.configure.base.cmaSound.get_squelch()
```

Enables the squelch function if you use the demodulator output as source. Impacts the AF1 OUT and AF2 OUT connectors if you configure it to use the demodulator signal.

return
enable: OFF | ON

get_volume() → int

```
# SCPI: CONFigure:BASE:CMASound:VOLume
value: int or bool = driver.configure.base.cmaSound.get_volume()
```

Configures the volume of the monitored CMA sound.

return
cma_sound: (integer or boolean) OFF Switches off the CMA sound without changing the volume setting ON Switches on the CMA sound without changing the volume setting number A number greater than zero sets the volume and switches on the CMA

sound. Zero sets the volume and switches off the CMA sound. Range: 0 % to 100 %, Unit: %

set_source(*sound_source: SoundSource*) → None

```
# SCPI: CONFigure:BASE:CMASound:SOURce
driver.configure.base.cmaSound.set_source(sound_source = enums.SoundSource.
↪AFONe)
```

Selects the audio source to be connected to the loudspeaker / headphones.

param sound_source
GENone | GENThree | AFONe | SPDif | DEModulator | LAN | AVIO GENone Generator 1 + generator 2 GENThree Generator 3 + generator 4 AFONe AF1 IN + AF2 IN SPDif SPDIF IN L + R DEModulator Demodulator output LAN LAN connector (voice over IP) AVIO Avionic generator

set_squelch(*enable: bool*) → None

```
# SCPI: CONFigure:BASE:CMASound:SQUelch
driver.configure.base.cmaSound.set_squelch(enable = False)
```

Enables the squelch function if you use the demodulator output as source. Impacts the AF1 OUT and AF2 OUT connectors if you configure it to use the demodulator signal.

param enable
OFF | ON

set_volume(*cma_sound: int*) → None

```
# SCPI: CONFigure:BASE:CMASound:VOLume
driver.configure.base.cmaSound.set_volume(cma_sound = 1)
```

Configures the volume of the monitored CMA sound.

param cma_sound
(integer or boolean) OFF Switches off the CMA sound without changing the volume setting ON Switches on the CMA sound without changing the volume setting number A number greater than zero sets the volume and switches on the CMA sound. Zero sets the volume and switches off the CMA sound. Range: 0 % to 100 %, Unit: %

6.3.2.6 Cprotection

SCPI Command:

```
CONFigure:BASE:CPRotection:RESet
```

class CprotectionCls

Cprotection commands group definition. 1 total commands, 0 Subgroups, 1 group commands

reset() → None

```
# SCPI: CONFigure:BASE:CPRotection:RESet
driver.configure.base.cprotection.reset()
```

Resets the protection circuit of the RF connectors.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:BASE:CPRotection:RESet
driver.configure.base.cprotection.reset_with_opc()
```

Resets the protection circuit of the RF connectors.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.3.2.7 Display

SCPI Command:

```
CONFigure:BASE:DISPlay:STATe
```

class DisplayCls

Display commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_state() → bool

```
# SCPI: CONFigure:BASE:DISPlay:STATe
value: bool = driver.configure.base.display.get_state()
```

Switches the instrument display off or on.

return
display_state: OFF | ON OFF: display off (black screen) ON: display on

set_state(*display_state: bool*) → None

```
# SCPI: CONFigure:BASE:DISPlay:STATe
driver.configure.base.display.set_state(display_state = False)
```

Switches the instrument display off or on.

param display_state
OFF | ON OFF: display off (black screen) ON: display on

6.3.2.8 Relay<Relay>

RepCap Settings

```
# Range: Ix1 .. Ix2
rc = driver.configure.base.relay.repcap_relay_get()
driver.configure.base.relay.repcap_relay_set(repcap.Relay.Ix1)
```

SCPI Command:

```
CONFigure:BASE:RELay<Index>
```

class RelayCls

Relay commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Relay, default value after init: Relay.Ix1

get(*relay=Relay.Default*) → Activity

```
# SCPI: CONFigure:BASE:RELay<Index>
value: enums.Activity = driver.configure.base.relay.get(relay = repcap.Relay.
↪Default)
```

Activates or deactivates relay 1 or 2 of the CONTROL connector.

param relay
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Relay')

return
activity: INACtive | ACTive

set(*activity: Activity*, *relay=Relay.Default*) → None

```
# SCPI: CONFigure:BASE:RELay<Index>
driver.configure.base.relay.set(activity = enums.Activity.ACTive, relay =␣
↪repcap.Relay.Default)
```

Activates or deactivates relay 1 or 2 of the CONTROL connector.

param activity
INACtive | ACTive

param relay
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Relay')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.base.relay.clone()
```

6.3.2.9 SysSound

SCPI Command:

```
CONFigure:BASE:SYSSound:VOLume
```

class SysSoundCls

SysSound commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_volume() → int

```
# SCPI: CONFigure:BASE:SYSSound:VOLume
value: int or bool = driver.configure.base.sysSound.get_volume()
```

Configures the volume of the system sound.

return
ssound: (integer or boolean) OFF Switches off the system sound without changing the volume setting ON Switches on the system sound without changing the volume setting number A number greater than zero sets the volume and switches on the system sound. Zero sets the volume and switches off the system sound. Range: 0 % to 100 %, Unit: %

set_volume(*ssound: int*) → None

```
# SCPI: CONFigure:BASE:SYSSound:VOLume
driver.configure.base.sysSound.set_volume(ssound = 1)
```

Configures the volume of the system sound.

param ssound
(integer or boolean) OFF Switches off the system sound without changing the volume setting ON Switches on the system sound without changing the volume setting number A number greater than zero sets the volume and switches on the system sound. Zero sets the volume and switches off the system sound. Range: 0 % to 100 %, Unit: %

6.3.2.10 Ttl<TTL>

RepCap Settings

```
# Range: Ix1 .. Ix2
rc = driver.configure.base.ttl.repcap_tTL_get()
driver.configure.base.ttl.repcap_tTL_set(repcap.TTL.Ix1)
```

SCPI Command:

```
CONFigure:BASE:TTL<Index>
```

class TtlCls

Ttl commands group definition. 3 total commands, 2 Subgroups, 1 group commands Repeated Capability: TTL, default value after init: TTL.Ix1

get(*tTL=TTL.Default*) → List[bool]

```
# SCPI: CONFigure:BASE:TTL<Index>
value: List[bool] = driver.configure.base.ttl.get(tTL = repcap.TTL.Default)
```

Sets or queries the individual bits of a TTL register of the CONTROL connector. A register with direction IN can only be queried. A register with direction OUT can be configured. Before querying the input register, update the values, see CONFigure:BASE:TTL2:UPDate.

param tTL
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Ttl')

return
pin_state: OFF | ON Comma-separated list of four values, one per bit of the register Register 1: Pin 1, 2, 3, 4 Register 2: Pin 14, 15, 16, 17 OFF = 0, ON = 1

set(*pin_state: List[bool]*, *tTL=TTL.Default*) → None

```
# SCPI: CONFigure:BASE:TTL<Index>
driver.configure.base.ttl.set(pin_state = [True, False, True], tTL = repcap.TTL.
↪Default)
```

Sets or queries the individual bits of a TTL register of the CONTROL connector. A register with direction IN can only be queried. A register with direction OUT can be configured. Before querying the input register, update the values, see CONFigure:BASE:TTL2:UPDate.

param pin_state
OFF | ON Comma-separated list of four values, one per bit of the register Register 1: Pin 1, 2, 3, 4 Register 2: Pin 14, 15, 16, 17 OFF = 0, ON = 1

param tTL
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Ttl')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.base.ttl.clone()
```

Subgroups

6.3.2.10.1 Direction

SCPI Command:

```
CONFigure:BASE:TTL<Index>:DIRection
```

class DirectionCls

Direction commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*tTL=TTL.Default*) → DirectionIo

```
# SCPI: CONFigure:BASE:TTL<Index>:DIRection
value: enums.DirectionIo = driver.configure.base.ttl.direction.get(tTL = repcap.
↪TTL.Default)
```

Configures the direction of a TTL register of the CONTROL connector. The direction of register 1 is fixed and can only be queried. The direction of register 2 can be configured.

param tTL
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Ttl')

return
direction: IN | OUT

set(*direction: DirectionIo*, *tTL=TTL.Default*) → None

```
# SCPI: CONFigure:BASE:TTL<Index>:DIRection
driver.configure.base.ttl.direction.set(direction = enums.DirectionIo.IN, tTL =␣
↪repcap.TTL.Default)
```

Configures the direction of a TTL register of the CONTROL connector. The direction of register 1 is fixed and can only be queried. The direction of register 2 can be configured.

param direction
IN | OUT

param tTL
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Ttl')

6.3.2.10.2 Update

SCPI Command:

```
CONFigure:BASE:TTL<Index>:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*tTL=TTL.Default*) → None

```
# SCPI: CONFigure:BASE:TTL<Index>:UPDate
driver.configure.base.ttl.update.set(tTL = repcap.TTL.Default)
```

If TTL2 is used in input direction, this command triggers the evaluation of the input signal and a refresh of the resulting bit values.

param tTL
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Ttl')

set_with_opc(*tTL=TTL.Default*, *opc_timeout_ms: int = -1*) → None

6.3.2.11 Zbox

SCPI Command:

```
CONFigure:BASE:ZBOX:ENABle
CONFigure:BASE:ZBOX:IMPedance
```

class ZboxCls

Zbox commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:BASE:ZBOX:ENABle
value: bool = driver.configure.base.zbox.get_enable()
```

No command help available

return
enable: No help available

get_impedance() → Impedance

```
# SCPI: CONFigure:BASE:ZBOX:IMPedance
value: enums.Impedance = driver.configure.base.zbox.get_impedance()
```

No command help available

return
impedance: No help available

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:BASE:ZBOX:ENABle
driver.configure.base.zbox.set_enable(enable = False)
```

No command help available

param enable
No help available

set_impedance(*impedance: Impedance*) → None

```
# SCPI: CONFigure:BASE:ZBOX:IMPedance
driver.configure.base.zbox.set_impedance(impedance = enums.Impedance.IHOL)
```

No command help available

param impedance
No help available

6.3.3 Display

SCPI Command:

```
CONFigure:DISPlay:TABSplit
```

class DisplayCls

Display commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_tab_split() → TabSplit

```
# SCPI: CONFigure:DISPlay:TABSplit
value: enums.TabSplit = driver.configure.display.get_tab_split()
```

Configures the tab mode of the GUI.

return
tab_split: TAB | SPLit TAB Merged mode, displaying a single tab at a time SPLit Split mode, displaying the generator tab on the left and the measurement tabs on the right

set_tab_split(*tab_split: TabSplit*) → None

```
# SCPI: CONFigure:DISPlay:TABSplit
driver.configure.display.set_tab_split(tab_split = enums.TabSplit.SPLit)
```

Configures the tab mode of the GUI.

param tab_split
TAB | SPLit TAB Merged mode, displaying a single tab at a time SPLit Split mode, displaying the generator tab on the left and the measurement tabs on the right

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.display.clone()
```

Subgroups

6.3.3.1 Application

SCPI Command:

```
CONFigure:DISPlay:APPLication:SELect
```

class ApplicationCls

Application commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → str

```
# SCPI: CONFigure:DISPlay:APPLication:SELect
value: str = driver.configure.display.application.get_select()
```

Selects the application to be displayed at the GUI. The command is useful as preparation for taking screenshots via remote commands. To make the GUI visible during remote control, use the command method RsCma.System.Display.update. To query a list of application selection strings, see method RsCma.Sense.Display.Applications.catalog.

return
current_app: String selecting the application

set_select(*current_app: str*) → None

```
# SCPI: CONFigure:DISPlay:APPLication:SELect
driver.configure.display.application.set_select(current_app = '1')
```

Selects the application to be displayed at the GUI. The command is useful as preparation for taking screenshots via remote commands. To make the GUI visible during remote control, use the command method RsCma.System.Display.update. To query a list of application selection strings, see method RsCma.Sense.Display.Applications.catalog.

param current_app
String selecting the application

6.3.4 GprfMeasurement

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:CREPetition
```

class GprfMeasurementCls

GprfMeasurement commands group definition. 121 total commands, 8 Subgroups, 1 group commands

get_crepetition() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CREPetition
value: bool = driver.configure.gprfMeasurement.get_crepetition()
```

Enables or disables the automatic configuration of the repetition mode. With enabled automatic configuration, the repetition mode of all measurements is set to 'Continuous' each time the instrument switches from remote operation to manual operation.

return
continuous_repetition: No help available

set_crepetition(*continuous_repetition: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:CREPetition
driver.configure.gprfMeasurement.set_crepetition(continuous_repetition = False)
```

Enables or disables the automatic configuration of the repetition mode. With enabled automatic configuration, the repetition mode of all measurements is set to 'Continuous' each time the instrument switches from remote operation to manual operation.

param continuous_repetition
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.clone()
```

Subgroups

6.3.4.1 Acp

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:ACP:MOEXception
CONFigure:GPRF:MEASurement<Instance>:ACP:STANdard
CONFigure:GPRF:MEASurement<Instance>:ACP:TOUT
CONFigure:GPRF:MEASurement<Instance>:ACP:REPetition
CONFigure:GPRF:MEASurement<Instance>:ACP:RCOupling
CONFigure:GPRF:MEASurement<Instance>:ACP:SCOunt
CONFigure:GPRF:MEASurement<Instance>:ACP:CSPace
```

(continues on next page)

(continued from previous page)

```
CONFigure:GPRF:MEASurement<Instance>:ACP:MBWidth
CONFigure:GPRF:MEASurement<Instance>:ACP:OFFSet
```

class AcpCls

Acp commands group definition. 16 total commands, 3 Subgroups, 9 group commands

get_cspace() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:CSPace
value: int = driver.configure.gprfMeasurement.acp.get_cspace()
```

Defines the channel spacing, that is the center frequency difference of two adjacent channels.

return
channel_space: Range: 100 Hz to 4 MHz, Unit: Hz

get_mbwidth() → List[int]

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:MBWidth
value: List[int] = driver.configure.gprfMeasurement.acp.get_mbwidth()
```

Defines the width of the measurement filter used to measure the channel power. The maximum allowed value is limited by the channel spacing, see method RsCma.Configure.GprfMeasurement.Acp.cspace.

return
meas_band_width: Range: 100 Hz to ChannelSpace, Unit: Hz

get_mo_exception() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:MOEXception
value: bool = driver.configure.gprfMeasurement.acp.get_mo_exception()
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

return
meas_on_exception: OFF | ON OFF Faulty results are rejected ON Results are never rejected

get_offset() → AcpOffset

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:OFFSet
value: enums.AcpOffset = driver.configure.gprfMeasurement.acp.get_offset()
```

Specifies an offset, moving the designated channel center frequency relative to the RF carrier center frequency.

return
offset: NONE | USB | LSB NONE No offset, for example for FM, PM, AM USB Positive offset, for USB modulation LSB Negative offset, for LSB modulation

get_rcoupling() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:RCOupling
value: bool = driver.configure.gprfMeasurement.acp.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupl: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:REPetition
value: enums.Repeat = driver.configure.gprfMeasurement.acp.get_repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:SCOunt
value: int = driver.configure.gprfMeasurement.acp.get_scount()
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval delivers one set of 'Current' results.

return
statistic_count: Range: 1 to 1000

get_standard() → StandardB

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:STANdard
value: enums.StandardB = driver.configure.gprfMeasurement.acp.get_standard()
```

Selects the digital standard of the measured signal.

return
standard: DMR | DPMR | NXDN | P25 | TETRa | LTE | CUSTom

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:TOUT
value: float = driver.configure.gprfMeasurement.acp.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_cspace(*channel_space: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:CSPace
driver.configure.gprfMeasurement.acp.set_cspace(channel_space = 1)
```

Defines the channel spacing, that is the center frequency difference of two adjacent channels.

param channel_space
Range: 100 Hz to 4 MHz, Unit: Hz

set_mbwidth(*meas_band_width: List[int]*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:MBWidth
driver.configure.gprfMeasurement.acp.set_mbwidth(meas_band_width = [1, 2, 3])
```

Defines the width of the measurement filter used to measure the channel power. The maximum allowed value is limited by the channel spacing, see method RsCma.Configure.GprfMeasurement.Acp.cspace.

param meas_band_width
Range: 100 Hz to ChannelSpace, Unit: Hz

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:MOEXception
driver.configure.gprfMeasurement.acp.set_mo_exception(meas_on_exception = False)
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

param meas_on_exception
OFF | ON OFF Faulty results are rejected ON Results are never rejected

set_offset(*offset: AcpOffset*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:OFFSet
driver.configure.gprfMeasurement.acp.set_offset(offset = enums.AcpOffset.LSB)
```

Specifies an offset, moving the designated channel center frequency relative to the RF carrier center frequency.

param offset
NONE | USB | LSB NONE No offset, for example for FM, PM, AM USB Positive offset, for USB modulation LSB Negative offset, for LSB modulation

set_rcoupling(*repetition_coupl: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:RCOupling
driver.configure.gprfMeasurement.acp.set_rcoupling(repetition_coupl = False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupl
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:REPetition
driver.configure.gprfMeasurement.acp.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:SCOunt
driver.configure.gprfMeasurement.acp.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval delivers one set of 'Current' results.

param statistic_count
Range: 1 to 1000

set_standard(*standard: StandardB*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:STANdard
driver.configure.gprfMeasurement.acp.set_standard(standard = enums.StandardB.
↪CUSTom)
```

Selects the digital standard of the measured signal.

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | CUSTom

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:TOUT
driver.configure.gprfMeasurement.acp.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.acp.clone()
```

Subgroups

6.3.4.1.1 Limit

class LimitCls

Limit 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.configure.gprfMeasurement.acp.limit.clone()
```

Subgroups

6.3.4.1.1.1 Aclr

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:ACLR
```

class AclrCls

Aclr commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class AclrStruct

Response structure. Fields:

- Limit_Ch_1: float: Upper ACLR limit for the channels '+1' and '-1' Range: -80 dB to 10 dB, Unit: dB
- Limit_Ch_2: float: Upper ACLR limit for the channels '+2' and '-2' Range: -80 dB to 10 dB, Unit: dB

get() → AclrStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:ACLR
value: AclrStruct = driver.configure.gprfMeasurement.acp.limit.aclr.get()
```

Configures upper limits for the measured ACLR values.

return
structure: for return value, see the help for AclrStruct structure arguments.

set(*limit_ch_1: float*, *limit_ch_2: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:ACLR
driver.configure.gprfMeasurement.acp.limit.aclr.set(limit_ch_1 = 1.0, limit_ch_
↪2 = 1.0)
```

Configures upper limits for the measured ACLR values.

param limit_ch_1
Upper ACLR limit for the channels '+1' and '-1' Range: -80 dB to 10 dB, Unit: dB

param limit_ch_2
Upper ACLR limit for the channels '+2' and '-2' Range: -80 dB to 10 dB, Unit: dB

6.3.4.1.1.2 Enable

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Enable_Ch_0: bool: OFF | ON Absolute power limit checks for the designated channel '0'
- Enable_Ch_1: bool: OFF | ON ACLR limit check for the neighbor channels '+1' and '-1'
- Enable_Ch_2: bool: OFF | ON ACLR limit check for the neighbor channels '+2' and '-2'

get() → EnableStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:ENABle
value: EnableStruct = driver.configure.gprfMeasurement.acp.limit.enable.get()
```

Enables or disables the ACLR and power limit checks.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*enable_ch_0: bool*, *enable_ch_1: bool*, *enable_ch_2: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:ENABle
driver.configure.gprfMeasurement.acp.limit.enable.set(enable_ch_0 = False,␣
↪enable_ch_1 = False, enable_ch_2 = False)
```

Enables or disables the ACLR and power limit checks.

param enable_ch_0
OFF | ON Absolute power limit checks for the designated channel '0'

param enable_ch_1
OFF | ON ACLR limit check for the neighbor channels '+1' and '-1'

param enable_ch_2
OFF | ON ACLR limit check for the neighbor channels '+2' and '-2'

6.3.4.1.1.3 Obw

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:OBW:ENABle
CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:OBW
```

class ObwCls

Obw commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:OBW:ENABle
value: bool = driver.configure.gprfMeasurement.acp.limit.obw.get_enable()
```

Enables or disables the OBW limit checks.

return
enable: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:OBW
value: float = driver.configure.gprfMeasurement.acp.limit.obw.get_value()
```

Configures an upper OBW limit.

return
limit: Range: 0 Hz to 8 MHz, Unit: Hz

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:OBW:ENABle
driver.configure.gprfMeasurement.acp.limit.obw.set_enable(enable = False)
```

Enables or disables the OBW limit checks.

param enable
OFF | ON

set_value(*limit: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:OBW
driver.configure.gprfMeasurement.acp.limit.obw.set_value(limit = 1.0)
```

Configures an upper OBW limit.

param limit
Range: 0 Hz to 8 MHz, Unit: Hz

6.3.4.1.1.4 Power

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:POWer
```

class PowerCls

Power commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PowerStruct

Response structure. Fields:

- Limit_Lower: float: Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm
- Limit_Upper: float: Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

get() → PowerStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:POWer
value: PowerStruct = driver.configure.gprfMeasurement.acp.limit.power.get()
```

Configures limits for the absolute power measured in the designated channel.

return
structure: for return value, see the help for PowerStruct structure arguments.

set(*limit_lower: float*, *limit_upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:LIMit:POWer
driver.configure.gprfMeasurement.acp.limit.power.set(limit_lower = 1.0, limit_
↪upper = 1.0)
```

Configures limits for the absolute power measured in the designated channel.

param limit_lower
Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm

param limit_upper
Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

6.3.4.1.2 Nxdn

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:ACP:NXDN:TRANsmission
```

class NxdnCls

Nxdn commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_transmission() → Transmission

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:NXDN:TRANsmission
value: enums.Transmission = driver.configure.gprfMeasurement.acp.nxdn.get_
↪transmission()
```

Queries the data rate in bits/s for enhanced full rate (EFR) or enhanced half rate (EHR) .

return
transmission: EHR4800 | EHR9600 | EFR9600

set_transmission(*transmission: Transmission*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:NXDN:TRANsmission
driver.configure.gprfMeasurement.acp.nxdn.set_transmission(transmission = enums.
↪Transmission.EFR9600)
```

Queries the data rate in bits/s for enhanced full rate (EFR) or enhanced half rate (EHR) .

param transmission
EHR4800 | EHR9600 | EFR9600

6.3.4.1.3 Obw

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:ACP:OBW:PERCentage
```

class ObwCls

Obw commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_percentage() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:OBW:PERCentage
value: float = driver.configure.gprfMeasurement.acp.obw.get_percentage()
```

Defines the power percentage to be used for calculation of the OBW results.

return
obw_percentage: Range: 70 % to 99.9 %, Unit: %

set_percentage(*obw_percentage: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:ACP:OBW:PERCentage
driver.configure.gprfMeasurement.acp.obw.set_percentage(obw_percentage = 1.0)
```

Defines the power percentage to be used for calculation of the OBW results.

param obw_percentage
Range: 70 % to 99.9 %, Unit: %

6.3.4.2 ExtPwrSensor

SCPI Command:

```
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:RCOupling
CONFigure:GPRF:MEASurement<Instance>:EPSensor:FREQuency
```

class ExtPwrSensorCls

ExtPwrSensor commands group definition. 9 total commands, 2 Subgroups, 6 group commands

get_frequency() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:FREQuency
value: float = driver.configure.gprfMeasurement.extPwrSensor.get_frequency()
```

Specifies the input frequency at the power sensor.

return
correction_freq: Range: Depends on used sensor model , Unit: Hz

get_rcoupling() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RCOupling
value: bool = driver.configure.gprfMeasurement.extPwrSensor.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupl: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:REPetition
value: enums.Repeat = driver.configure.gprfMeasurement.extPwrSensor.get_
↪repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

get_resolution() → ExtSensorResolution

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RESolution
value: enums.ExtSensorResolution = driver.configure.gprfMeasurement.
↪extPwrSensor.get_resolution()
```

Defines the number of decimal places of the power results displayed in the graphical user interface. This command does not affect results queried via remote control commands.

return
resolution: PD0 | PD1 | PD2 | PD3 PDn Results rounded to n places after the decimal point

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:SCOunt
value: int = driver.configure.gprfMeasurement.extPwrSensor.get_scount()
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval comprises one power result requested from the power sensor.

return
statistic_count: Range: 1 to 1000

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:TOUT
value: float = driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_frequency(*correction_freq: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:FREQuency
driver.configure.gprfMeasurement.extPwrSensor.set_frequency(correction_freq = 1.
↪0)
```

Specifies the input frequency at the power sensor.

param correction_freq
Range: Depends on used sensor model , Unit: Hz

set_rcoupling(*repetition_coupl: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RCOupling
driver.configure.gprfMeasurement.extPwrSensor.set_rcoupling(repetition_coupl =␣
↪False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupl
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:REPetition
driver.configure.gprfMeasurement.extPwrSensor.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

set_resolution(*resolution: ExtSensorResolution*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:RESolution
driver.configure.gprfMeasurement.extPwrSensor.set_resolution(resolution = enums.
↪ExtSensorResolution.PD0)
```

Defines the number of decimal places of the power results displayed in the graphical user interface. This command does not affect results queried via remote control commands.

param resolution
PD0 | PD1 | PD2 | PD3 PDn Results rounded to n places after the decimal point

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:SCOunt
driver.configure.gprfMeasurement.extPwrSensor.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval comprises one power result requested from the power sensor.

param statistic_count
Range: 1 to 1000

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:TOUT
driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.extPwrSensor.clone()
```

Subgroups

6.3.4.2.1 Attenuation

SCPI Command:

```
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.gprfMeasurement.extPwrSensor.attenuation.get_
↪state()
```

Specifies whether there is an attenuator or amplifier between the power sensor and the DUT.

return
attenuator_state: OFF | ON OFF Direct connection to DUT ON Attenuator or amplifier between power sensor and DUT. Configure also the attenuation, see method RsCma.Configure.GprfMeasurement.ExtPwrSensor.Attenuation.value.

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation
value: float = driver.configure.gprfMeasurement.extPwrSensor.attenuation.get_
↪value()
```

Specifies the attenuation or gain of a component between the power sensor and the DUT. The power readings are corrected accordingly. Configure also the attenuation state, see method RsCma.Configure.GprfMeasurement.ExtPwrSensor.Attenuation. state.

return
attenuation: Range: -50 dB to 50 dB, Unit: dB

set_state(*attenuator_state: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation:STATe
driver.configure.gprfMeasurement.extPwrSensor.attenuation.set_state(attenuator_
↪state = False)
```

Specifies whether there is an attenuator or amplifier between the power sensor and the DUT.

param attenuator_state
OFF | ON OFF Direct connection to DUT ON Attenuator or amplifier between power sensor and DUT. Configure also the attenuation, see method RsCma.Configure.GprfMeasurement.ExtPwrSensor.Attenuation.value.

set_value(*attenuation: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:ATTenuation
driver.configure.gprfMeasurement.extPwrSensor.attenuation.set_value(attenuation␣
↪= 1.0)
```

Specifies the attenuation or gain of a component between the power sensor and the DUT. The power readings are corrected accordingly. Configure also the attenuation state, see method RsCma.Configure.GprfMeasurement.ExtPwrSensor.Attenuation. state.

param attenuation
Range: -50 dB to 50 dB, Unit: dB

6.3.4.2.2 Average

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:APERture
```

class AverageCls

Average commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_aperture() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:APERture
value: float = driver.configure.gprfMeasurement.extPwrSensor.average.get_
↪aperture()
```

Defines the size of the acquisition interval.

return
aperture: Range: 10E-6 s to 0.3 s, Unit: s

set_aperture(*aperture: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:EPSensor:AVERage:APERture
driver.configure.gprfMeasurement.extPwrSensor.average.set_aperture(aperture = 1.
↪0)
```

Defines the size of the acquisition interval.

param aperture
Range: 10E-6 s to 0.3 s, Unit: s

6.3.4.3 FftSpecAn

SCPI Command:

```
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:RCOupling
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
```

class FftSpecAnCls

FftSpecAn commands group definition. 13 total commands, 2 Subgroups, 9 group commands

get_amode() → AveragingMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:AMODe
value: enums.AveragingMode = driver.configure.gprfMeasurement.fftSpecAn.get_
↪amode()
```

Defines how the average FFT trace is derived from the current trace.

return
averaging_mode: LINear | LOGarithmic LINear Averaging of the linear powers LOGarithmic Averaging of the dBm values

get_detector() → DetectorSimple

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:DETector
value: enums.DetectorSimple = driver.configure.gprfMeasurement.fftSpecAn.get_
↪detector()
```

Defines how a spectrum diagram point is calculated from adjacent frequency domain samples.

return
detector: PEAK | RMS PEAK The sample with the largest power is displayed. RMS The RMS value of the samples is displayed.

get_fft_length() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FFTLength
value: int = driver.configure.gprfMeasurement.fftSpecAn.get_fft_length()
```

Specifies the number of samples used for the FFT analysis.

return
length: You can enter values between 1024 and 16384. The setting is rounded to the closest integer power of two.

get_fspan() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FSPan
value: float = driver.configure.gprfMeasurement.fftSpecAn.get_fspan()
```

Defines the frequency range to be measured and thus the x-axis of the spectrum result diagram.

return
frequency_span: You can enter values between 10 kHz and 20 MHz. The setting is rounded to the closest of the following values: 10 / 20 / 40 / 80 / 160 / 320 / 640 kHz 1.25 / 2.5 / 5 / 10 / 20 MHz Unit: Hz

get_mo_exception() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MOEXception
value: bool = driver.configure.gprfMeasurement.fftSpecAn.get_mo_exception()
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

return
meas_on_exception: OFF | ON OFF Faulty results are rejected ON Results are never rejected

get_rcoupling() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:RCOupling
value: bool = driver.configure.gprfMeasurement.fftSpecAn.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupl: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:REPetition
value: enums.Repeat = driver.configure.gprfMeasurement.fftSpecAn.get_
↪repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
value: int = driver.configure.gprfMeasurement.fftSpecAn.get_scount()
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval comprises the number of samples defined by the 'FFT Length'.

return
statistic_count: Range: 1 to 1000

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
value: float = driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_amode(*averaging_mode: AveragingMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:AMODe
driver.configure.gprfMeasurement.fftSpecAn.set_amode(averaging_mode = enums.
↪AveragingMode.LINear)
```

Defines how the average FFT trace is derived from the current trace.

param averaging_mode
LINear | LOGarithmic LINear Averaging of the linear powers LOGarithmic Averaging of the dBm values

set_detector(*detector: DetectorSimple*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:DETector
driver.configure.gprfMeasurement.fftSpecAn.set_detector(detector = enums.
↪DetectorSimple.PEAK)
```

Defines how a spectrum diagram point is calculated from adjacent frequency domain samples.

param detector
PEAK | RMS PEAK The sample with the largest power is displayed. RMS The RMS value of the samples is displayed.

set_fft_length(*length: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FFTLength
driver.configure.gprfMeasurement.fftSpecAn.set_fft_length(length = 1)
```

Specifies the number of samples used for the FFT analysis.

param length
You can enter values between 1024 and 16384. The setting is rounded to the closest integer power of two.

set_fspan(*frequency_span: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:FSPan
driver.configure.gprfMeasurement.fftSpecAn.set_fspan(frequency_span = 1.0)
```

Defines the frequency range to be measured and thus the x-axis of the spectrum result diagram.

param frequency_span
You can enter values between 10 kHz and 20 MHz. The setting is rounded to the closest of the following values: 10 / 20 / 40 / 80 / 160 / 320 / 640 kHz 1.25 / 2.5 / 5 / 10 / 20 MHz Unit: Hz

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MOEXception
driver.configure.gprfMeasurement.fftSpecAn.set_mo_exception(meas_on_exception =␣
↪False)
```

Specifies whether measurement results that the CMA identifies as faulty or inaccurate are rejected.

param meas_on_exception
OFF | ON OFF Faulty results are rejected ON Results are never rejected

set_rcoupling(*repetition_coupl: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:RCOupling
driver.configure.gprfMeasurement.fftSpecAn.set_rcoupling(repetition_coupl =␣
↪False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupl
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:REPetition
driver.configure.gprfMeasurement.fftSpecAn.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:SCOunt
driver.configure.gprfMeasurement.fftSpecAn.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval comprises the number of samples defined by the 'FFT Length'.

param statistic_count
Range: 1 to 1000

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.fftSpecAn.clone()
```

Subgroups

6.3.4.3.1 Marker<MarkerOther>

RepCap Settings

```
# Range: Nr2 .. Nr5
rc = driver.configure.gprfMeasurement.fftSpecAn.marker.repcap_markerOther_get()
driver.configure.gprfMeasurement.fftSpecAn.marker.repcap_markerOther_set(repcap.
↪MarkerOther.Nr2)
```

class MarkerCls

Marker commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: MarkerOther, default value after init: MarkerOther.Nr2

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.fftSpecAn.marker.clone()
```

Subgroups

6.3.4.3.1.1 Enable

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer:ENABle:ALL
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set_all(*enable: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer:ENABle:ALL
driver.configure.gprfMeasurement.fftSpecAn.marker.enable.set_all(enable = False)
```

Enables or disables the markers R, 2 and 3.

param enable
OFF | ON

6.3.4.3.1.2 Placement

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>:PLACement
```

class PlacementCls

Placement commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*markerOther=MarkerOther.Default*) → MarkerPlacement

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>:PLACement
value: enums.MarkerPlacement = driver.configure.gprfMeasurement.fftSpecAn.
↪marker.placement.get(markerOther = repcap.MarkerOther.Default)
```

Selects between absolute coordinates and delta coordinates relative to the reference marker, for marker number <no>.

param markerOther
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Marker')

return
placement: ABSolute | RELative

set(*placement: MarkerPlacement*, *markerOther=MarkerOther.Default*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>:PLACement
driver.configure.gprfMeasurement.fftSpecAn.marker.placement.set(placement =␣
↪enums.MarkerPlacement.ABSolute, markerOther = repcap.MarkerOther.Default)
```

Selects between absolute coordinates and delta coordinates relative to the reference marker, for marker number <no>.

param placement
ABSolute | RELative

param markerOther
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Marker')

6.3.4.3.2 PeakSearch

SCPI Command:

```
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 Configures marker 0 OFF Search from PeakRangeFrom0 to PeakRangeTo0 ON Search the full frequency span
- Peak_Range_From_0: float: Start of peak search range for FullSpanEnable0 = OFF Range: - span / 2 to span / 2, Unit: Hz
- Peak_Range_To_0: float: End of peak search range for FullSpanEnable0 = OFF Range: - span / 2 to span / 2, Unit: Hz
- Full_Span_Enable_1: bool: Configures marker 1
- Peak_Range_From_1: float: No parameter help available
- Peak_Range_To_1: float: No parameter help available
- Full_Span_Enable_2: bool: Configures marker 2
- Peak_Range_From_2: float: No parameter help available
- Peak_Range_To_2: float: No parameter help available
- Full_Span_Enable_3: bool: Configures marker 3
- Peak_Range_From_3: float: No parameter help available
- Peak_Range_To_3: float: No parameter help available
- Full_Span_Enable_4: bool: Configures marker 4
- Peak_Range_From_4: float: No parameter help available
- Peak_Range_To_4: float: No parameter help available

get_noa_markers() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch:NOAMarkers
value: int = driver.configure.gprfMeasurement.fftSpecAn.peakSearch.get_noa_
↪markers()
```

Specifies the number of active peak search markers.

return
no_active_markers: Range: 0 to 5

get_value() → ValueStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch
value: ValueStruct = driver.configure.gprfMeasurement.fftSpecAn.peakSearch.get_
↪value()
```

Defines the peak search ranges for the five markers. In this command, the markers are labeled 0 to 4. At the graphical user interface, they are labeled 1 to 5. The allowed ranges, reset values and default units are identical for all markers. The ranges depend on the configured span, see method RsCma.Configure.GprfMeasurement.FftSpecAn.fspan.

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.gprfMeasurement.fftSpecAn.peakSearch.set_noa_markers(no_active_
↪markers = 1)
```

Specifies the number of active peak search markers.

param no_active_markers
Range: 0 to 5

set_value(*value: ValueStruct*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:FFTSanalyzer:PSEarch
structure = driver.configure.gprfMeasurement.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.gprfMeasurement.fftSpecAn.peakSearch.set_value(value =␣
↪structure)
```

Defines the peak search ranges for the five markers. In this command, the markers are labeled 0 to 4. At the graphical user interface, they are labeled 1 to 5. The allowed ranges, reset values and default units are identical for all markers. The ranges depend on the configured span, see method RsCma.Configure.GprfMeasurement.FftSpecAn.fspan.

param value
see the help for ValueStruct structure arguments.

6.3.4.4 IqRecorder

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOFFset
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:SAMPles
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:IQFile
CONFigure:GPRF:MEASurement<Instance>:IQRecorder:WTFile
```

class IqRecorderCls

IqRecorder commands group definition. 12 total commands, 2 Subgroups, 8 group commands

get_format_py() → IqFormat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FORMat
value: enums.IqFormat = driver.configure.gprfMeasurement.iqRecorder.get_format_
↪py()
```

Selects a coordinate system for representation of the measurement results.

return
format_py: IQ | RPHI IQ Cartesian coordinates (I- and Q-axis) RPHI Polar coordinates (radius and angle)

get_iq_file() → str

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQFile
value: str = driver.configure.gprfMeasurement.iqRecorder.get_iq_file()
```

Selects the name and path of the result file. The results are stored in the file in binary format. To write the file, start the measurement via INITiate:GPRF:MEAS:IQRecorder ON.

return
iq_save_file: String parameter to specify the file name and path The supported file name extensions are .iqw and .wv. The extension selects the file type.

get_munit() → MagnitudeUnit

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MUNit
value: enums.MagnitudeUnit = driver.configure.gprfMeasurement.iqRecorder.get_
↪munit()
```

Selects a physical unit for representation of the measured I/Q amplitudes.

return
magnitude_unit: VOLT | RAW Voltage or raw I/Q data relative to full scale

get_ratio() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:RATio
value: float = driver.configure.gprfMeasurement.iqRecorder.get_ratio()
```

Defines the sample ratio and thus the sample rate (<sample rate> = <sample ratio> * <max sample rate>) .

return
ratio: Range: 0.1 to 1

get_samples() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:SAMPles
value: int = driver.configure.gprfMeasurement.iqRecorder.get_samples()
```

No command help available

return
samples: No help available

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
value: float = driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

get_toffset() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOFFset
value: int = driver.configure.gprfMeasurement.iqRecorder.get_toffset()
```

No command help available

return
trigger_offset: No help available

get_wt_file() → FileSave

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:WTFile
value: enums.FileSave = driver.configure.gprfMeasurement.iqRecorder.get_wt_
↪file()
```

Selects whether the results are written to an I/Q file, to the memory or both. For file selection, see method RsCma. Configure.GprfMeasurement.IqRecorder.iqFile.

return
save_to_iq_file: OFF | ON | ONLY OFF The results are only stored in the memory. They can be queried via remote control commands. ON The results are stored in the memory and in a file. ONLY The results are only stored in a file. Use this selection if you want to record huge amounts of data that do not fit into the memory.

set_format_py(*format_py: IqFormat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FORMat
driver.configure.gprfMeasurement.iqRecorder.set_format_py(format_py = enums.
↪IqFormat.IQ)
```

Selects a coordinate system for representation of the measurement results.

param format_py
IQ | RPHI IQ Cartesian coordinates (I- and Q-axis) RPHI Polar coordinates (radius and angle)

set_iq_file(*iq_save_file: str*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:IQFile
driver.configure.gprfMeasurement.iqRecorder.set_iq_file(iq_save_file = '1')
```

Selects the name and path of the result file. The results are stored in the file in binary format. To write the file, start the measurement via INITiate:GPRF:MEAS:IQRecorder ON.

param iq_save_file
String parameter to specify the file name and path The supported file name extensions are .iqw and .wv. The extension selects the file type.

set_munit(*magnitude_unit: MagnitudeUnit*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:MUNit
driver.configure.gprfMeasurement.iqRecorder.set_munit(magnitude_unit = enums.
↪MagnitudeUnit.RAW)
```

Selects a physical unit for representation of the measured I/Q amplitudes.

param magnitude_unit
VOLT | RAW Voltage or raw I/Q data relative to full scale

set_ratio(*ratio: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:RATio
driver.configure.gprfMeasurement.iqRecorder.set_ratio(ratio = 1.0)
```

Defines the sample ratio and thus the sample rate (<sample rate> = <sample ratio> * <max sample rate>) .

param ratio
Range: 0.1 to 1

set_samples(*samples: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:SAMPles
driver.configure.gprfMeasurement.iqRecorder.set_samples(samples = 1)
```

No command help available

param samples
No help available

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOUT
driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

set_toffset(*trigger_offset: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:TOFFset
driver.configure.gprfMeasurement.iqRecorder.set_toffset(trigger_offset = 1)
```

No command help available

param trigger_offset
No help available

set_wt_file(*save_to_iq_file: FileSave*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:WTFile
driver.configure.gprfMeasurement.iqRecorder.set_wt_file(save_to_iq_file = enums.
↪FileSave.OFF)
```

Selects whether the results are written to an I/Q file, to the memory or both. For file selection, see method RsCma. Configure.GprfMeasurement.IqRecorder.iqFile.

param save_to_iq_file
OFF | ON | ONLY OFF The results are only stored in the memory. They can be queried via remote control commands. ON The results are stored in the memory and in a file. ONLY The results are only stored in a file. Use this selection if you want to record huge amounts of data that do not fit into the memory.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.iqRecorder.clone()
```

Subgroups

6.3.4.4.1 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: Samples before the trigger event Range: 1 to 67108863
- Capt_Samp_Aft_Trig: int: Samples after the trigger event Range: 1 to 67108863

get() → CaptureStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:CAPTure
value: CaptureStruct = driver.configure.gprfMeasurement.iqRecorder.capture.get()
```

Defines the number of samples to be evaluated before the trigger event and after the trigger event. The maximum total number of samples is 67108864. The sum of the two settings must not exceed this value.

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.gprfMeasurement.iqRecorder.capture.set(capt_samp_bef_trig = 1,␣
↪capt_samp_aft_trig = 1)
```

Defines the number of samples to be evaluated before the trigger event and after the trigger event. The maximum total number of samples is 67108864. The sum of the two settings must not exceed this value.

param capt_samp_bef_trig
Samples before the trigger event Range: 1 to 67108863

param capt_samp_aft_trig
Samples after the trigger event Range: 1 to 67108863

6.3.4.4.2 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.gprfMeasurement.iqRecorder.
↪filterPy.get_type_py()
```

Selects the IF filter type.

return
filter_type: BANDpass | GAUSs BANDpass Bandpass filter with selectable bandwidth
GAUSs Gaussian filter with selectable bandwidth

set_type_py(*filter_type: RbwFilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:TYPE
driver.configure.gprfMeasurement.iqRecorder.filterPy.set_type_py(filter_type =␣
↪enums.RbwFilterType.BANDpass)
```

Selects the IF filter type.

param filter_type
BANDpass | GAUSs BANDpass Bandpass filter with selectable bandwidth GAUSs Gaussian filter with selectable bandwidth

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.iqRecorder.filterPy.clone()
```

Subgroups

6.3.4.4.2.1 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.gprfMeasurement.iqRecorder.filterPy.bandpass.
↪get_bandwidth()
```

Selects the bandwidth for the bandpass filter.

return
bandpass_bw: You can enter values between 1 kHz and 40 MHz. The setting is rounded to the closest of the following values: 1 kHz / 10 kHz / 100 kHz / 1 MHz / 10 MHz / 40 MHz Unit: Hz

set_bandwidth(*bandpass_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:BANDpass:BWIDth
driver.configure.gprfMeasurement.iqRecorder.filterPy.bandpass.set_
↪bandwidth(bandpass_bw = 1.0)
```

Selects the bandwidth for the bandpass filter.

param bandpass_bw
You can enter values between 1 kHz and 40 MHz. The setting is rounded to the closest of the following values: 1 kHz / 10 kHz / 100 kHz / 1 MHz / 10 MHz / 40 MHz Unit: Hz

6.3.4.4.2.2 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.gprfMeasurement.iqRecorder.filterPy.gauss.get_
↪bandwidth()
```

Selects the bandwidth for the Gaussian filter.

return
gauss_bw: You can enter values between 1 kHz and 10 MHz. The setting is rounded to the closest of the following values: 1 kHz / 10 kHz / 100 kHz / 1 MHz / 10 MHz Unit: Hz

set_bandwidth(*gauss_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:IQRecorder:FILTer:GAUSs:BWIDth
driver.configure.gprfMeasurement.iqRecorder.filterPy.gauss.set_bandwidth(gauss_
↪bw = 1.0)
```

Selects the bandwidth for the Gaussian filter.

param gauss_bw
You can enter values between 1 kHz and 10 MHz. The setting is rounded to the closest of the following values: 1 kHz / 10 kHz / 100 kHz / 1 MHz / 10 MHz Unit: Hz

6.3.4.5 Nrt

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:SCOunt
CONFigure:GPRF:MEASurement<Instance>:NRT:REPetition
CONFigure:GPRF:MEASurement<Instance>:NRT:RCOupling
CONFigure:GPRF:MEASurement<Instance>:NRT:FREQuency
CONFigure:GPRF:MEASurement<Instance>:NRT:CCDF
CONFigure:GPRF:MEASurement<Instance>:NRT:BWIDth
CONFigure:GPRF:MEASurement<Instance>:NRT:RESolution
CONFigure:GPRF:MEASurement<Instance>:NRT:DIRection
CONFigure:GPRF:MEASurement<Instance>:NRT:PEPHoldtime
CONFigure:GPRF:MEASurement<Instance>:NRT:DEVice
```

class NrtCls

Nrt commands group definition. 25 total commands, 3 Subgroups, 10 group commands

get_bandwidth() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:BWIDth
value: float = driver.configure.gprfMeasurement.nrt.get_bandwidth()
```

Sets the video filter bandwidth for the rectified RF signal.

return
bandwidth: The entered value is rounded to the nearest of the following values: 4 kHz | 200 kHz | 600 kHz Range: 4000 Hz to 600 kHz, Unit: Hz

get_cumulative_distrib_fnc() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:CCDF
value: float = driver.configure.gprfMeasurement.nrt.get_cumulative_distrib_fnc()
```

Configures a PEP threshold for calculation of the CCDF result. Note the default value dBm. To enter watts, append W to the value, for example 2W. To query watts, append a W to your query: CONFigure:GPRF:MEAS:NRT:CCDF? W.

return
threshold: Range: 1 W to 300 W, Unit: dBm

get_device() → NrtDevice

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:DEVice
value: enums.NrtDevice = driver.configure.gprfMeasurement.nrt.get_device()
```

Selects the used power sensor model.

return
device: N14 | N43 | N44 N14: R&S NRT-Z14 N43: R&S NRT-Z43 N44: R&S NRT-Z44

get_direction() → PowerSignalDirection

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:DIRection
value: enums.PowerSignalDirection = driver.configure.gprfMeasurement.nrt.get_
↪direction()
```

Defines the forward direction relative to the ports of the power sensor.

return
direction: FWD | REV | AUTO FWD The forward direction is fixed from port 1 to port 2. REV The forward direction is fixed from port 2 to port 1. AUTO The forward direction is selected automatically.

get_frequency() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FREQuency
value: float = driver.configure.gprfMeasurement.nrt.get_frequency()
```

Specifies the input frequency at the power sensor.

return
correction_freq: Range: Depends on the power sensor model , Unit: Hz

get_pep_hold_time() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:PEPHoldtime
value: float = driver.configure.gprfMeasurement.nrt.get_pep_hold_time()
```

Sets the hold time for the measurement of the peak envelope power.

return
pep_hold_time: Range: 1E-3 s to 0.1 s, Unit: s

get_rcoupling() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:RCOupling
value: bool = driver.configure.gprfMeasurement.nrt.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupl: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REPetition
value: enums.Repeat = driver.configure.gprfMeasurement.nrt.get_repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

get_resolution() → LowHigh

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:RESolution
value: enums.LowHigh = driver.configure.gprfMeasurement.nrt.get_resolution()
```

Selects the measurement resolution.

return
bandwidth: LOW | HIGH

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:SCOunt
value: int = driver.configure.gprfMeasurement.nrt.get_scount()
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval comprises one set of results for both directions.

return
statistic_count: Range: 1 to 1000

set_bandwidth(*bandwidth: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:BWIDth
driver.configure.gprfMeasurement.nrt.set_bandwidth(bandwidth = 1.0)
```

Sets the video filter bandwidth for the rectified RF signal.

param bandwidth
The entered value is rounded to the nearest of the following values: 4 kHz | 200 kHz | 600 kHz Range: 4000 Hz to 600 kHz, Unit: Hz

set_cumulative_distrib_fnc(*threshold: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:CCDF
driver.configure.gprfMeasurement.nrt.set_cumulative_distrib_fnc(threshold = 1.0)
```

Configures a PEP threshold for calculation of the CCDF result. Note the default value dBm. To enter watts, append W to the value, for example 2W. To query watts, append a W to your query: CONFigure:GPRF:MEAS:NRT:CCDF? W.

param threshold
Range: 1 W to 300 W, Unit: dBm

set_device(*device: NrtDevice*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:DEVice
driver.configure.gprfMeasurement.nrt.set_device(device = enums.NrtDevice.N14)
```

Selects the used power sensor model.

param device
N14 | N43 | N44 N14: R&S NRT-Z14 N43: R&S NRT-Z43 N44: R&S NRT-Z44

set_direction(*direction: PowerSignalDirection*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:DIRection
driver.configure.gprfMeasurement.nrt.set_direction(direction = enums.
↪PowerSignalDirection.AUTO)
```

Defines the forward direction relative to the ports of the power sensor.

param direction
FWD | REV | AUTO FWD The forward direction is fixed from port 1 to port 2. REV The forward direction is fixed from port 2 to port 1. AUTO The forward direction is selected automatically.

set_frequency(*correction_freq: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FREQuency
driver.configure.gprfMeasurement.nrt.set_frequency(correction_freq = 1.0)
```

Specifies the input frequency at the power sensor.

param correction_freq
Range: Depends on the power sensor model , Unit: Hz

set_pep_hold_time(*pep_hold_time: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:PEPHoldtime
driver.configure.gprfMeasurement.nrt.set_pep_hold_time(pep_hold_time = 1.0)
```

Sets the hold time for the measurement of the peak envelope power.

param pep_hold_time
Range: 1E-3 s to 0.1 s, Unit: s

set_rcoupling(*repetition_coupl: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:RCOupling
driver.configure.gprfMeasurement.nrt.set_rcoupling(repetition_coupl = False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupl
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REPetition
driver.configure.gprfMeasurement.nrt.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

set_resolution(*bandwidth: LowHigh*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:RESolution
driver.configure.gprfMeasurement.nrt.set_resolution(bandwidth = enums.LowHigh.
↪HIGH)
```

Selects the measurement resolution.

param bandwidth
LOW | HIGH

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:SCOunt
driver.configure.gprfMeasurement.nrt.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval comprises one set of results for both directions.

param statistic_count
Range: 1 to 1000

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.nrt.clone()
```

Subgroups

6.3.4.5.1 Attenuation

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation:PORT
CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation:STATe
CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation
```

class AttenuationCls

Attenuation commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_port() → AttenuationPort

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation:PORT
value: enums.AttenuationPort = driver.configure.gprfMeasurement.nrt.attenuation.
↪get_port()
```

Selects the NRT-Z port to be used as measurement point.

return
attenuation_port: SOURce | LOAD

get_state() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation:STATe
value: bool = driver.configure.gprfMeasurement.nrt.attenuation.get_state()
```

Enables or disables the compensation of the external attenuation configured via method RsCma.Configure.GprfMeasurement. Nrt.Attenuation.value.

return
attenuator_state: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation
value: float = driver.configure.gprfMeasurement.nrt.attenuation.get_value()
```

Specifies the attenuation of a component between the power sensor and the DUT, typically a cable. The power readings are corrected accordingly, if the correction is enabled via method RsCma.Configure.GprfMeasurement.Nrt.Attenuation.state.

return
attenuation: Range: 0 dB to 100 dB, Unit: dB

set_port(*attenuation_port: AttenuationPort*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation:PORT
driver.configure.gprfMeasurement.nrt.attenuation.set_port(attenuation_port =␣
↪enums.AttenuationPort.LOAD)
```

Selects the NRT-Z port to be used as measurement point.

param attenuation_port
SOURce | LOAD

set_state(*attenuator_state: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation:STATe
driver.configure.gprfMeasurement.nrt.attenuation.set_state(attenuator_state =␣
↪False)
```

Enables or disables the compensation of the external attenuation configured via method RsCma.Configure.GprfMeasurement. Nrt.Attenuation.value.

param attenuator_state
OFF | ON

set_value(*attenuation: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:ATTenuation
driver.configure.gprfMeasurement.nrt.attenuation.set_value(attenuation = 1.0)
```

Specifies the attenuation of a component between the power sensor and the DUT, typically a cable. The power readings are corrected accordingly, if the correction is enabled via method RsCma.Configure.GprfMeasurement.Nrt.Attenuation.state.

param attenuation
Range: 0 dB to 100 dB, Unit: dB

6.3.4.5.2 Forward

class ForwardCls

Forward 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.configure.gprfMeasurement.nrt.forward.clone()
```

Subgroups

6.3.4.5.2.1 Limit

class LimitCls

Limit commands group definition. 5 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.nrt.forward.limit.clone()
```

Subgroups

6.3.4.5.2.2 Cfactor

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:CFACtor
```

class CfactorCls

Cfactor commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CfactorStruct

Response structure. Fields:

- Lower: float: Range: 0 dB to 50 dB, Unit: dB
- Upper: float: Range: 0 dB to 50 dB, Unit: dB

get() → CfactorStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:CFACtor
value: CfactorStruct = driver.configure.gprfMeasurement.nrt.forward.limit.
↪cfactor.get()
```

Configures limits for the crest factor results.

return
structure: for return value, see the help for CfactorStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:CFACtor
driver.configure.gprfMeasurement.nrt.forward.limit.cfactor.set(lower = 1.0,␣
↪upper = 1.0)
```

Configures limits for the crest factor results.

param lower
Range: 0 dB to 50 dB, Unit: dB

param upper
Range: 0 dB to 50 dB, Unit: dB

6.3.4.5.2.3 CumulativeDistribFnc

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:CCDF
```

class CumulativeDistribFncCls

CumulativeDistribFnc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CumulativeDistribFncStruct

Response structure. Fields:

- Lower: float: Range: 0 % to 100 %, Unit: %

- Upper: float: Range: 0 % to 100 %, Unit: %

get() → CumulativeDistribFncStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:CCDF
value: CumulativeDistribFncStruct = driver.configure.gprfMeasurement.nrt.
↪forward.limit.cumulativeDistribFnc.get()
```

Configures limits for the CCDF results.

return
structure: for return value, see the help for CumulativeDistribFncStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:CCDF
driver.configure.gprfMeasurement.nrt.forward.limit.cumulativeDistribFnc.
↪set(lower = 1.0, upper = 1.0)
```

Configures limits for the CCDF results.

param lower
Range: 0 % to 100 %, Unit: %

param upper
Range: 0 % to 100 %, Unit: %

6.3.4.5.2.4 Enable

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Power: bool: OFF | ON
- Pep: bool: OFF | ON
- Crest_Factor: bool: OFF | ON
- Ccdf: bool: OFF | ON

get() → EnableStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:ENABle
value: EnableStruct = driver.configure.gprfMeasurement.nrt.forward.limit.enable.
↪get()
```

Enables/disables the limit check for the forward direction results.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*power: bool*, *pep: bool*, *crest_factor: bool*, *ccdf: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:ENABle
driver.configure.gprfMeasurement.nrt.forward.limit.enable.set(power = False,␣
↪pep = False, crest_factor = False, ccdf = False)
```

Enables/disables the limit check for the forward direction results.

param power
OFF | ON

param pep
OFF | ON

param crest_factor
OFF | ON

param ccdf
OFF | ON

6.3.4.5.2.5 Pep

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:PEP
```

class PepCls

Pep commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PepStruct

Response structure. Fields:

- Lower: float: Range: Depends on sensor model , Unit: dBm
- Upper: float: Range: Depends on sensor model , Unit: dBm

get() → PepStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:PEP
value: PepStruct = driver.configure.gprfMeasurement.nrt.forward.limit.pep.get()
```

Configures limits for the PEP results.

return
structure: for return value, see the help for PepStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:PEP
driver.configure.gprfMeasurement.nrt.forward.limit.pep.set(lower = 1.0, upper =␣
↪1.0)
```

Configures limits for the PEP results.

param lower
Range: Depends on sensor model , Unit: dBm

param upper
Range: Depends on sensor model , Unit: dBm

6.3.4.5.2.6 Power

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:POWer
```

class PowerCls

Power commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PowerStruct

Response structure. Fields:

- Lower: float: Range: Depends on sensor model , Unit: dBm
- Upper: float: Range: Depends on sensor model , Unit: dBm

get() → PowerStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:POWer
value: PowerStruct = driver.configure.gprfMeasurement.nrt.forward.limit.power.
↪get()
```

Configures limits for the forward power results.

return
structure: for return value, see the help for PowerStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:LIMit:POWer
driver.configure.gprfMeasurement.nrt.forward.limit.power.set(lower = 1.0, upper␣
↪= 1.0)
```

Configures limits for the forward power results.

param lower
Range: Depends on sensor model , Unit: dBm

param upper
Range: Depends on sensor model , Unit: dBm

6.3.4.5.2.7 Value

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:VALue:ENABle
```

class ValueCls

Value commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → DirPwrSensorFwdValue

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:VALue:ENABle
value: enums.DirPwrSensorFwdValue = driver.configure.gprfMeasurement.nrt.
↪forward.value.get_enable()
```

Selects the forward result to be measured.

return
value: FPWR | PEP | CFAC | CCDF FPWR Forward power PEP Peak envelope power CFAC Crest factor CCDF Complementary cumulative distribution function

set_enable(*value: DirPwrSensorFwdValue*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:FWARd:VALue:ENABle
driver.configure.gprfMeasurement.nrt.forward.value.set_enable(value = enums.
↪DirPwrSensorFwdValue.CCDF)
```

Selects the forward result to be measured.

param value
FPWR | PEP | CFAC | CCDF FPWR Forward power PEP Peak envelope power CFAC Crest factor CCDF Complementary cumulative distribution function

6.3.4.5.3 Reverse

class ReverseCls

Reverse 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.configure.gprfMeasurement.nrt.reverse.clone()
```

Subgroups

6.3.4.5.3.1 Limit

class LimitCls

Limit commands group definition. 5 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.nrt.reverse.limit.clone()
```

Subgroups

6.3.4.5.3.2 Enable

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Power: bool: OFF | ON
- Return_Loss: bool: OFF | ON
- Reflection: bool: OFF | ON
- Swr: bool: OFF | ON

get() → EnableStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:ENABle
value: EnableStruct = driver.configure.gprfMeasurement.nrt.reverse.limit.enable.
↪get()
```

Enables/disables the limit check for the reverse direction results.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*power: bool*, *return_loss: bool*, *reflection: bool*, *swr: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:ENABle
driver.configure.gprfMeasurement.nrt.reverse.limit.enable.set(power = False,␣
↪return_loss = False, reflection = False, swr = False)
```

Enables/disables the limit check for the reverse direction results.

param power
OFF | ON

param return_loss
OFF | ON

param reflection
OFF | ON

param swr
OFF | ON

6.3.4.5.3.3 Power

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:POWer
```

class PowerCls

Power commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PowerStruct

Response structure. Fields:

- Lower: float: Range: Depends on sensor model , Unit: dBm
- Upper: float: Range: Depends on sensor model , Unit: dBm

get() → PowerStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:POWer
value: PowerStruct = driver.configure.gprfMeasurement.nrt.reverse.limit.power.
↪get()
```

Configures limits for the reverse power results.

return
structure: for return value, see the help for PowerStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:POWer
driver.configure.gprfMeasurement.nrt.reverse.limit.power.set(lower = 1.0, upper␣
↪= 1.0)
```

Configures limits for the reverse power results.

param lower
Range: Depends on sensor model , Unit: dBm

param upper
Range: Depends on sensor model , Unit: dBm

6.3.4.5.3.4 Reflection

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:REFLection
```

class ReflectionCls

Reflection commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ReflectionStruct

Response structure. Fields:

- Lower: float: Range: 0 % to 100 %, Unit: %
- Upper: float: Range: 0 % to 100 %, Unit: %

get() → ReflectionStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:REFLection
value: ReflectionStruct = driver.configure.gprfMeasurement.nrt.reverse.limit.
↪reflection.get()
```

Configures limits for the reflection results.

return
structure: for return value, see the help for ReflectionStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:REFLection
driver.configure.gprfMeasurement.nrt.reverse.limit.reflection.set(lower = 1.0,␣
↪upper = 1.0)
```

Configures limits for the reflection results.

param lower
Range: 0 % to 100 %, Unit: %

param upper
Range: 0 % to 100 %, Unit: %

6.3.4.5.3.5 Rloss

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:RLOSs
```

class RlossCls

Rloss commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RlossStruct

Response structure. Fields:

- Lower: float: Range: 0 dB to 50 dB, Unit: dB
- Upper: float: Range: 0 dB to 50 dB, Unit: dB

get() → RlossStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:RLOSs
value: RlossStruct = driver.configure.gprfMeasurement.nrt.reverse.limit.rloss.
↪get()
```

Configures limits for the return loss results.

return
structure: for return value, see the help for RlossStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:RLOSs
driver.configure.gprfMeasurement.nrt.reverse.limit.rloss.set(lower = 1.0, upper␣
↪= 1.0)
```

Configures limits for the return loss results.

param lower
Range: 0 dB to 50 dB, Unit: dB

param upper
Range: 0 dB to 50 dB, Unit: dB

6.3.4.5.3.6 Swr

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:SWR
```

class SwrCls

Swr commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SwrStruct

Response structure. Fields:

- Lower: float: Range: 1 to 50
- Upper: float: Range: 1 to 50

get() → SwrStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:SWR
value: SwrStruct = driver.configure.gprfMeasurement.nrt.reverse.limit.swr.get()
```

Configures limits for the SWR results.

return
structure: for return value, see the help for SwrStruct structure arguments.

set(*lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:LIMit:SWR
driver.configure.gprfMeasurement.nrt.reverse.limit.swr.set(lower = 1.0, upper =␣
↪1.0)
```

Configures limits for the SWR results.

param lower
Range: 1 to 50

param upper
Range: 1 to 50

6.3.4.5.3.7 Value

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:VALue:ENABle
```

class ValueCls

Value commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → DirPwrSensorRevValue

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:VALue:ENABle
value: enums.DirPwrSensorRevValue = driver.configure.gprfMeasurement.nrt.
↪reverse.value.get_enable()
```

Selects the reverse result to be measured.

INTRO_CMD_HELP: The effect of the value RPWR depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: RPWR selects the reverse power result
- CFAC or CCDF: RPWR selects the forward power result

return

value: RPWR | RLOS | REFL | SWR RPWR Reverse power or forward power RLOS Return loss REFL Reflection SWR Standing wave ratio

set_enable(*value: DirPwrSensorRevValue*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:NRT:REVerse:VALue:ENABle
driver.configure.gprfMeasurement.nrt.reverse.value.set_enable(value = enums.
↪DirPwrSensorRevValue.REFL)
```

Selects the reverse result to be measured.

INTRO_CMD_HELP: The effect of the value RPWR depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: RPWR selects the reverse power result
- CFAC or CCDF: RPWR selects the forward power result

param value

RPWR | RLOS | REFL | SWR RPWR Reverse power or forward power RLOS Return loss REFL Reflection SWR Standing wave ratio

6.3.4.6 Power

SCPI Command:

```
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:RCOupling
CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
```

class PowerCls

Power commands group definition. 9 total commands, 1 Subgroups, 6 group commands

get_mlength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MLENgth
value: float = driver.configure.gprfMeasurement.power.get_mlength()
```

Sets the time interval used to calculate one set of 'Current' power result (RMS value, minimum and maximum) . The maximum allowed value is limited by the step length, see method RsCma.Configure.GprfMeasurement.Power.slength.

return
meas_length: Range: 10E-6 s to StepLength, Unit: s

get_rcoupling() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:RCOupling
value: bool = driver.configure.gprfMeasurement.power.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupl: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:REPetition
value: enums.Repeat = driver.configure.gprfMeasurement.power.get_repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
value: int = driver.configure.gprfMeasurement.power.get_scount()
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval delivers one 'Current' power result.

return
statistic_count: Range: 1 to 100E+3

get_slength() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SLENgth
value: float = driver.configure.gprfMeasurement.power.get_slength()
```

Sets the time between the beginning of two consecutive measurement intervals.

return
step_length: Range: 50E-6 s to 1 s, Unit: s

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:TOUT
value: float = driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_mlength(*meas_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:MLENgth
driver.configure.gprfMeasurement.power.set_mlength(meas_length = 1.0)
```

Sets the time interval used to calculate one set of 'Current' power result (RMS value, minimum and maximum) . The maximum allowed value is limited by the step length, see method RsCma.Configure.GprfMeasurement.Power.slength.

param meas_length
Range: 10E-6 s to StepLength, Unit: s

set_rcoupling(*repetition_coupl: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:RCOupling
driver.configure.gprfMeasurement.power.set_rcoupling(repetition_coupl = False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupl
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:REPetition
driver.configure.gprfMeasurement.power.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SCOunt
driver.configure.gprfMeasurement.power.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval delivers one 'Current' power result.

param statistic_count
Range: 1 to 100E+3

set_slength(*step_length: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:SLENgth
driver.configure.gprfMeasurement.power.set_slength(step_length = 1.0)
```

Sets the time between the beginning of two consecutive measurement intervals.

param step_length
Range: 50E-6 s to 1 s, Unit: s

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:TOUT
driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.power.clone()
```

Subgroups

6.3.4.6.1 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() → FilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:TYPE
value: enums.FilterType = driver.configure.gprfMeasurement.power.filterPy.get_
↪type_py()
```

Selects the IF filter type.

return
filter_type: BANDpass | GAUSs BANDpass Bandpass filter with selectable bandwidth GAUSs Gaussian filter with selectable bandwidth

set_type_py(*filter_type: FilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:TYPE
driver.configure.gprfMeasurement.power.filterPy.set_type_py(filter_type = enums.
↪FilterType.BANDpass)
```

Selects the IF filter type.

param filter_type
BANDpass | GAUSs BANDpass Bandpass filter with selectable bandwidth GAUSs Gaussian filter with selectable bandwidth

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.power.filterPy.clone()
```

Subgroups

6.3.4.6.1.1 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.gprfMeasurement.power.filterPy.bandpass.get_
↪bandwidth()
```

Selects the bandwidth for the bandpass filter.

return
bandpass_bw: You can enter values between 1 kHz and 20 MHz. The setting is rounded to the closest of the following values: 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 1.08 / 2 / 2.7 / 3 / 4.5 / 5 / 9 / 10 / 13.5 / 18 / 20 / 40 MHz Unit: Hz

set_bandwidth(*bandpass_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:BANDpass:BWIDth
driver.configure.gprfMeasurement.power.filterPy.bandpass.set_bandwidth(bandpass_
↪bw = 1.0)
```

Selects the bandwidth for the bandpass filter.

param bandpass_bw
You can enter values between 1 kHz and 20 MHz. The setting is rounded to the closest of the following values: 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 1.08 / 2 / 2.7 / 3 / 4.5 / 5 / 9 / 10 / 13.5 / 18 / 20 / 40 MHz Unit: Hz

6.3.4.6.1.2 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.gprfMeasurement.power.filterPy.gauss.get_
↪bandwidth()
```

Selects the bandwidth for the Gaussian filter.

return
gauss_bw: You can enter values between 10 Hz and 10 MHz. The setting is rounded to the closest of the following values: 10 / 30 / 50 / 100 / 300 / 500 Hz 1 / 3 / 5 / 6.25 / 10 / 30 / 50 / 100 / 300 / 500 kHz 1 / 3 / 5 / 10 MHz Unit: Hz

set_bandwidth(*gauss_bw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:POWer:FILTer:GAUSs:BWIDth
driver.configure.gprfMeasurement.power.filterPy.gauss.set_bandwidth(gauss_bw =␣
↪1.0)
```

Selects the bandwidth for the Gaussian filter.

param gauss_bw
You can enter values between 10 Hz and 10 MHz. The setting is rounded to the closest of the following values: 10 / 30 / 50 / 100 / 300 / 500 Hz 1 / 3 / 5 / 6.25 / 10 / 30 / 50 / 100 / 300 / 500 kHz 1 / 3 / 5 / 10 MHz Unit: Hz

6.3.4.7 RfSettings

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:RFSettings:CONNector
CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
CONFigure:GPRF:MEASurement<Instance>:RFSettings:ENPower
CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
CONFigure:GPRF:MEASurement<Instance>:RFSettings:RFCoupling
```

class RfSettingsCls

RfSettings commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_connector() → InputConnector

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:CONNector
value: enums.InputConnector = driver.configure.gprfMeasurement.rfSettings.get_
↪connector()
```

Selects the input connector for the measured RF signal.

return
input_connector: RFCom | RFIN

get_eattenuation() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
value: float = driver.configure.gprfMeasurement.rfSettings.get_eattenuation()
```

Specifies the external attenuation in the input path. Negative values specify a gain.

return
rf_input_ext_att: Range: -50 dB to 90 dB, Unit: dB

get_envelope_power() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:ENPower
value: float = driver.configure.gprfMeasurement.rfSettings.get_envelope_power()
```

Sets the expected nominal power of the measured RF signal. The allowed range depends on several other settings, for example on the selected connector and the external attenuation. For supported ranges, refer to the data sheet.

return
exp_nominal_power: Unit: dBm

get_frequency() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
value: float = driver.configure.gprfMeasurement.rfSettings.get_frequency()
```

Sets the center frequency of the RF analyzer.

return
analyzer_freq: Range: 100 kHz to 3 GHz, Unit: Hz

get_rf_coupling() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:RFCoupling
value: bool = driver.configure.gprfMeasurement.rfSettings.get_rf_coupling()
```

Couples the frequency setting of the measurement to the corresponding generator setting.

return
rf_coupling: OFF | ON

set_connector(*input_connector: InputConnector*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:CONNector
driver.configure.gprfMeasurement.rfSettings.set_connector(input_connector =␣
↪enums.InputConnector.RFCom)
```

Selects the input connector for the measured RF signal.

param input_connector
RFCom | RFIN

set_eattenuation(*rf_input_ext_att: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:EATTenuation
driver.configure.gprfMeasurement.rfSettings.set_eattenuation(rf_input_ext_att =␣
↪1.0)
```

Specifies the external attenuation in the input path. Negative values specify a gain.

param rf_input_ext_att
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.gprfMeasurement.rfSettings.set_envelope_power(exp_nominal_
↪power = 1.0)
```

Sets the expected nominal power of the measured RF signal. The allowed range depends on several other settings, for example on the selected connector and the external attenuation. For supported ranges, refer to the data sheet.

param exp_nominal_power
Unit: dBm

set_frequency(*analyzer_freq: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:FREQuency
driver.configure.gprfMeasurement.rfSettings.set_frequency(analyzer_freq = 1.0)
```

Sets the center frequency of the RF analyzer.

param analyzer_freq
Range: 100 kHz to 3 GHz, Unit: Hz

set_rf_coupling(*rf_coupling: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:RFSettings:RFCoupling
driver.configure.gprfMeasurement.rfSettings.set_rf_coupling(rf_coupling = False)
```

Couples the frequency setting of the measurement to the corresponding generator setting.

param rf_coupling
OFF | ON

6.3.4.8 Spectrum

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:RCOupling
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
```

class SpectrumCls

Spectrum commands group definition. 31 total commands, 6 Subgroups, 5 group commands

get_amode() → AveragingMode

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
value: enums.AveragingMode = driver.configure.gprfMeasurement.spectrum.get_
↪amode()
```

Defines how the average trace is derived from the current trace.

return
averaging_mode: LINear | LOGarithmic LINear Averaging of the linear powers LOGarithmic Averaging of the dBm values

get_rcoupling() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:RCOupling
value: bool = driver.configure.gprfMeasurement.spectrum.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupl: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
value: enums.Repeat = driver.configure.gprfMeasurement.spectrum.get_repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

get_scount() → int

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
value: int = driver.configure.gprfMeasurement.spectrum.get_scount()
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval covers the frequency span defined for the 'Frequency Sweep' mode, or the sweep time defined for the 'Zero Span' mode.

return
statistic_count: Range: 1 to 1000

get_timeout() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
value: float = driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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: Unit: s

set_amode(*averaging_mode: AveragingMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:AMODe
driver.configure.gprfMeasurement.spectrum.set_amode(averaging_mode = enums.
↪AveragingMode.LINear)
```

Defines how the average trace is derived from the current trace.

param averaging_mode
LINear | LOGarithmic LINear Averaging of the linear powers LOGarithmic Averaging of the dBm values

set_rcoupling(*repetition_coupl: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:RCOupling
driver.configure.gprfMeasurement.spectrum.set_rcoupling(repetition_coupl =␣
↪False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupl
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:REPetition
driver.configure.gprfMeasurement.spectrum.set_repetition(repetition = enums.
↪Repeat.CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after one measurement cycle CONTinuous Continuous measurement, running until explicitly terminated

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:SCOunt
driver.configure.gprfMeasurement.spectrum.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval covers the frequency span defined for the 'Frequency Sweep' mode, or the sweep time defined for the 'Zero Span' mode.

param statistic_count
Range: 1 to 1000

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TOUT
driver.configure.gprfMeasurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. 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
Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.spectrum.clone()
```

Subgroups

6.3.4.8.1 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.gprfMeasurement.spectrum.freqSweep.clone()
```

Subgroups

6.3.4.8.1.1 Rbw

SCPI Command:

```
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.gprfMeasurement.spectrum.freqSweep.rbw.get_auto()
```

Enables or disables automatic configuration of the resolution bandwidth (RBW) for the frequency sweep mode.

return
rbw_auto: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW
value: float = driver.configure.gprfMeasurement.spectrum.freqSweep.rbw.get_
↪value()
```

Selects the bandwidth of the Gaussian resolution filter for the frequency sweep mode. Setting this value is only possible if the automatic RBW selection is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.FreqSweep.Rbw.auto.

return
rbw: You can enter values between 100 Hz and 10 MHz. The setting is rounded to the closest of the following values: 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz Unit: Hz

set_auto(*rbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW:AUTO
driver.configure.gprfMeasurement.spectrum.freqSweep.rbw.set_auto(rbw_auto =␣
↪False)
```

Enables or disables automatic configuration of the resolution bandwidth (RBW) for the frequency sweep mode.

param rbw_auto
OFF | ON

set_value(*rbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:RBW
driver.configure.gprfMeasurement.spectrum.freqSweep.rbw.set_value(rbw = 1.0)
```

Selects the bandwidth of the Gaussian resolution filter for the frequency sweep mode. Setting this value is only possible if the automatic RBW selection is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.FreqSweep.Rbw.auto.

param rbw
You can enter values between 100 Hz and 10 MHz. The setting is rounded to the closest of the following values: 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz Unit: Hz

6.3.4.8.1.2 Swt

SCPI Command:

```
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.gprfMeasurement.spectrum.freqSweep.swt.get_auto()
```

Enables or disables automatic configuration of the sweep time for the frequency sweep mode.

return
sweep_time_auto: OFF | ON

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT
value: float = driver.configure.gprfMeasurement.spectrum.freqSweep.swt.get_
↪value()
```

Specifies the sweep time for the frequency sweep mode. Setting this value is only possible if the automatic configuration is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.FreqSweep.Swt.auto.

return
sweep_time: Range: 3.99602E-3 s to 2000 s, Unit: s

set_auto(*sweep_time_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:SWT:AUTO
driver.configure.gprfMeasurement.spectrum.freqSweep.swt.set_auto(sweep_time_
↪auto = False)
```

Enables or disables automatic configuration of the sweep time for the 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.gprfMeasurement.spectrum.freqSweep.swt.set_value(sweep_time =␣
↪1.0)
```

Specifies the sweep time for the frequency sweep mode. Setting this value is only possible if the automatic configuration is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.FreqSweep.Swt.auto.

param sweep_time
Range: 3.99602E-3 s to 2000 s, Unit: s

6.3.4.8.1.3 Vbw

SCPI Command:

```
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.gprfMeasurement.spectrum.freqSweep.vbw.get_auto()
```

Enables or disables automatic configuration of the video bandwidth (VBW) for the 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.gprfMeasurement.spectrum.freqSweep.vbw.
↪get_value()
```

Selects the bandwidth of the video filter for the frequency sweep mode. Setting this value is only possible if the automatic VBW selection is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.FreqSweep.Vbw.auto.

return
vbw: (float or boolean) You can enter values between 10 Hz and 10 MHz. The setting is rounded to the closest of the following values: 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz OFF disables the filter Unit: Hz

set_auto(*vbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW:AUTO
driver.configure.gprfMeasurement.spectrum.freqSweep.vbw.set_auto(vbw_auto =␣
↪False)
```

Enables or disables automatic configuration of the video bandwidth (VBW) for the frequency sweep mode.

param vbw_auto
OFF | ON

set_value(*vbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:VBW
driver.configure.gprfMeasurement.spectrum.freqSweep.vbw.set_value(vbw = 1.0)
```

Selects the bandwidth of the video filter for the frequency sweep mode. Setting this value is only possible if the automatic VBW selection is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.FreqSweep.Vbw.auto.

param vbw
(float or boolean) You can enter values between 10 Hz and 10 MHz. The setting is rounded to the closest of the following values: 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz OFF disables the filter Unit: Hz

6.3.4.8.2 Frequency

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:CENTer
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STARt
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
```

class FrequencyCls

Frequency commands group definition. 7 total commands, 2 Subgroups, 3 group commands

get_center() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:CENTer
value: float = driver.configure.gprfMeasurement.spectrum.frequency.get_center()
```

Specifies the center frequency of the measurement. You can also configure this setting via method RsCma.Configure. GprfMeasurement.RfSettings.frequency.

return
center_frequency: Range: 100 kHz to 3 GHz, Unit: Hz

get_start() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STARt
value: float = driver.configure.gprfMeasurement.spectrum.frequency.get_start()
```

Specifies the start frequency of the measured span for the frequency sweep mode.

return
start_frequency: Range: 99500 Hz to 2.9999995 GHz, Unit: Hz

get_stop() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
value: float = driver.configure.gprfMeasurement.spectrum.frequency.get_stop()
```

Specifies the stop frequency of the measured span for the frequency sweep mode.

return
stop_frequency: Range: 100500 Hz to 3.0000005 GHz, Unit: Hz

set_center(*center_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:CENTer
driver.configure.gprfMeasurement.spectrum.frequency.set_center(center_frequency␣
↪= 1.0)
```

Specifies the center frequency of the measurement. You can also configure this setting via method RsCma.Configure. GprfMeasurement.RfSettings.frequency.

param center_frequency
Range: 100 kHz to 3 GHz, Unit: Hz

set_start(*start_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STARt
driver.configure.gprfMeasurement.spectrum.frequency.set_start(start_frequency =␣
↪1.0)
```

Specifies the start frequency of the measured span for the frequency sweep mode.

param start_frequency
Range: 99500 Hz to 2.9999995 GHz, Unit: Hz

set_stop(*stop_frequency: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:STOP
driver.configure.gprfMeasurement.spectrum.frequency.set_stop(stop_frequency = 1.
↪0)
```

Specifies the stop frequency of the measured span for the frequency sweep mode.

param stop_frequency
Range: 100500 Hz to 3.0000005 GHz, Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.spectrum.frequency.clone()
```

Subgroups

6.3.4.8.2.1 Marker

class MarkerCls

Marker commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.spectrum.frequency.marker.clone()
```

Subgroups

6.3.4.8.2.2 Placement

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:PLACement
```

class PlacementCls

Placement commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*markerOther=MarkerOther.Nr2*) → MarkerPlacement

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>
↪:PLACement
value: enums.MarkerPlacement = driver.configure.gprfMeasurement.spectrum.
↪frequency.marker.placement.get(markerOther = repcap.MarkerOther.Nr2)
```

Selects between absolute coordinates and delta coordinates relative to the reference marker, for marker number <no> and frequency sweep mode.

param markerOther
optional repeated capability selector. Default value: Nr2

return
placement: ABSolute | RELative

set(*placement: MarkerPlacement*, *markerOther=MarkerOther.Nr2*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>
↪:PLACement
driver.configure.gprfMeasurement.spectrum.frequency.marker.placement.
↪set(placement = enums.MarkerPlacement.ABSolute, markerOther = repcap.
↪MarkerOther.Nr2)
```

Selects between absolute coordinates and delta coordinates relative to the reference marker, for marker number <no> and frequency sweep mode.

param placement
ABSolute | RELative

param markerOther
optional repeated capability selector. Default value: Nr2

6.3.4.8.2.3 Range

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RangeStruct

Response structure. Fields:

- Xrange_Lower: float: Unit: Hz
- Xrange_Upper: float: Unit: Hz

get(*marker=Marker.Nr1*) → RangeStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:RANGe
value: RangeStruct = driver.configure.gprfMeasurement.spectrum.frequency.marker.
↪range.get(marker = repcap.Marker.Nr1)
```

Specifies a peak search range, for marker number <no> and frequency sweep mode. Marker number one is the reference marker. The allowed range of frequency values corresponds to the measured span.

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for RangeStruct structure arguments.

set(*xrange_lower: float*, *xrange_upper: float*, *marker=Marker.Nr1*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:RANGe
driver.configure.gprfMeasurement.spectrum.frequency.marker.range.set(xrange_
↪lower = 1.0, xrange_upper = 1.0, marker = repcap.Marker.Nr1)
```

Specifies a peak search range, for marker number <no> and frequency sweep mode. Marker number one is the reference marker. The allowed range of frequency values corresponds to the measured span.

param xrange_lower
Unit: Hz

param xrange_upper
Unit: Hz

param marker
optional repeated capability selector. Default value: Nr1

6.3.4.8.2.4 Span

SCPI Command:

```
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.gprfMeasurement.spectrum.frequency.
↪span.get_mode()
```

Selects the measurement mode.

return
span_mode: FSWeep | ZSPan FSWeep Frequency sweep mode ZSPan Zero span mode

get_value() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN
value: float = driver.configure.gprfMeasurement.spectrum.frequency.span.get_
↪value()
```

Specifies the frequency span for the frequency sweep mode.

return
frequency_span: Range: 1000 Hz to 2.999901 GHz, Unit: Hz

set_mode(*span_mode: SpanMode*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN:MODE
driver.configure.gprfMeasurement.spectrum.frequency.span.set_mode(span_mode =␣
↪enums.SpanMode.FSWeep)
```

Selects the measurement mode.

param span_mode
FSWeep | ZSPan FSWeep Frequency sweep mode ZSPan Zero span mode

set_value(*frequency_span: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:SPAN
driver.configure.gprfMeasurement.spectrum.frequency.span.set_value(frequency_
↪span = 1.0)
```

Specifies the frequency span for the frequency sweep mode.

param frequency_span
Range: 1000 Hz to 2.999901 GHz, Unit: Hz

6.3.4.8.3 Marker

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:MARKer:DETector
```

class MarkerCls

Marker commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_detector() → Detector

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:MARKer:DETector
value: enums.Detector = driver.configure.gprfMeasurement.spectrum.marker.get_
↪detector()
```

Selects the detector used to calculate the 1001 values of the result traces from the raw set of samples.

return
detector: AVERage | RMS | SAMPle | MINPeak | MAXPeak | AUTopeak

set_detector(*detector: Detector*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:MARKer:DETector
driver.configure.gprfMeasurement.spectrum.marker.set_detector(detector = enums.
↪Detector.AUTopeak)
```

Selects the detector used to calculate the 1001 values of the result traces from the raw set of samples.

param detector
AVERage | RMS | SAMPle | MINPeak | MAXPeak | AUTopeak

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.spectrum.marker.clone()
```

Subgroups

6.3.4.8.3.1 Enable

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:MARKer:ENABle:ALL
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set_all(*enable: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:MARKer:ENABle:ALL
driver.configure.gprfMeasurement.spectrum.marker.enable.set_all(enable = False)
```

Enables or disables all markers for the frequency sweep mode and the zero span mode.

param enable
OFF | ON

6.3.4.8.4 Tgenerator

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:ENABle
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:NORMalize
```

class TgeneratorCls

Tgenerator commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:ENABle
value: bool = driver.configure.gprfMeasurement.spectrum.tgenerator.get_enable()
```

Enables the tracking mode, so that the generator application acts as tracking generator.

return
state: OFF | ON

get_normalize() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:NORMalize
value: bool = driver.configure.gprfMeasurement.spectrum.tgenerator.get_
↪normalize()
```

Enables the normalization of the frequency sweep results for measurements with tracking generator.

return
normalize: OFF | ON

set_enable(*state: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:ENABle
driver.configure.gprfMeasurement.spectrum.tgenerator.set_enable(state = False)
```

Enables the tracking mode, so that the generator application acts as tracking generator.

param state
OFF | ON

set_normalize(*normalize: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:NORMalize
driver.configure.gprfMeasurement.spectrum.tgenerator.set_normalize(normalize =␣
↪False)
```

Enables the normalization of the frequency sweep results for measurements with tracking generator.

param normalize
OFF | ON

6.3.4.8.5 Vswr

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:VSWR:MODE
```

class VswrCls

Vswr commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → bool

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:VSWR:MODE
value: bool = driver.configure.gprfMeasurement.spectrum.vswr.get_mode()
```

Enables the 'VSWR Mode' to measure the VSWR with CMA tracking generator.

return
vswr_mode: OFF | ON

set_mode(*vswr_mode: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:VSWR:MODE
driver.configure.gprfMeasurement.spectrum.vswr.set_mode(vswr_mode = False)
```

Enables the 'VSWR Mode' to measure the VSWR with CMA tracking generator.

param vswr_mode
OFF | ON

6.3.4.8.6 ZeroSpan

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
```

class ZeroSpanCls

ZeroSpan commands group definition. 8 total commands, 3 Subgroups, 1 group commands

get_swt() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
value: float = driver.configure.gprfMeasurement.spectrum.zeroSpan.get_swt()
```

Specifies the sweep time for the zero span mode.

return
sweep_time: Range: 500.5E-6 s to 2000 s, Unit: s

set_swt(*sweep_time: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:SWT
driver.configure.gprfMeasurement.spectrum.zeroSpan.set_swt(sweep_time = 1.0)
```

Specifies the sweep time for the zero span mode.

param sweep_time
Range: 500.5E-6 s to 2000 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.spectrum.zeroSpan.clone()
```

Subgroups

6.3.4.8.6.1 Marker

class MarkerCls

Marker commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.gprfMeasurement.spectrum.zeroSpan.marker.clone()
```

Subgroups

6.3.4.8.6.2 Placement

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:PLACement
```

class PlacementCls

Placement commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*markerOther=MarkerOther.Nr2*) → MarkerPlacement

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:PLACement
value: enums.MarkerPlacement = driver.configure.gprfMeasurement.spectrum.
↪zeroSpan.marker.placement.get(markerOther = repcap.MarkerOther.Nr2)
```

Selects between absolute coordinates and coordinates relative to the reference marker, for marker number <no> and zero span mode.

param markerOther
optional repeated capability selector. Default value: Nr2

return
placement: ABSolute | RELative

set(*placement: MarkerPlacement*, *markerOther=MarkerOther.Nr2*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:PLACement
driver.configure.gprfMeasurement.spectrum.zeroSpan.marker.placement.
↪set(placement = enums.MarkerPlacement.ABSolute, markerOther = repcap.
↪MarkerOther.Nr2)
```

Selects between absolute coordinates and coordinates relative to the reference marker, for marker number <no> and zero span mode.

param placement
ABSolute | RELative

param markerOther
optional repeated capability selector. Default value: Nr2

6.3.4.8.6.3 Range

SCPI Command:

```
CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RangeStruct

Response structure. Fields:

- Xrange_Lower: float: Range: 0 s to SweepTime, Unit: s
- Xrange_Upper: float: Range: 0 s to SweepTime, Unit: s

get(*marker=Marker.Nr1*) → RangeStruct

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:RANGe
value: RangeStruct = driver.configure.gprfMeasurement.spectrum.zeroSpan.marker.
↪range.get(marker = repcap.Marker.Nr1)
```

Specifies the peak search range, for marker number <no> and zero span mode. Marker number one is the reference marker.

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for RangeStruct structure arguments.

set(*xrange_lower: float*, *xrange_upper: float*, *marker=Marker.Nr1*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:RANGe
driver.configure.gprfMeasurement.spectrum.zeroSpan.marker.range.set(xrange_
↪lower = 1.0, xrange_upper = 1.0, marker = repcap.Marker.Nr1)
```

Specifies the peak search range, for marker number <no> and zero span mode. Marker number one is the reference marker.

param xrange_lower
Range: 0 s to SweepTime, Unit: s

param xrange_upper
Range: 0 s to SweepTime, Unit: s

param marker
optional repeated capability selector. Default value: Nr1

6.3.4.8.6.4 Rbw

SCPI Command:

```
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.gprfMeasurement.spectrum.zeroSpan.rbw.get_
↪bandpass()
```

Sets the resolution bandwidth of the 'Bandpass' filter type in the spectrum measurement. The resolution bandwidth is fixed in this software version.

return
rbw_bandpass: Range: 40 MHz to 40 MHz, Unit: Hz

get_gauss() → float

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:GAUSs
value: float = driver.configure.gprfMeasurement.spectrum.zeroSpan.rbw.get_
↪gauss()
```

Selects the bandwidth of the Gaussian resolution filter for the zero span mode. To use this filter, configure also method RsCma.Configure.GprfMeasurement.Spectrum.ZeroSpan.Rbw.typePy.

return
rbw: You can enter values between 100 Hz and 10 MHz. The setting is rounded to the closest of the following values: 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz Unit: Hz

get_type_py() → RbwFilterType

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:TYPE
value: enums.RbwFilterType = driver.configure.gprfMeasurement.spectrum.zeroSpan.
↪rbw.get_type_py()
```

Selects the resolution filter type for the zero span mode.

return
rbw_type: GAUSs | BANDpass GAUSs Gaussian filter with configurable bandwidth BANDpass Bandpass filter with 40 MHz bandwidth

set_bandpass(*rbw_bandpass: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:BANDpass
driver.configure.gprfMeasurement.spectrum.zeroSpan.rbw.set_bandpass(rbw_
↪bandpass = 1.0)
```

Sets the resolution bandwidth of the 'Bandpass' filter type in the spectrum measurement. The resolution bandwidth is fixed in this software version.

param rbw_bandpass
Range: 40 MHz to 40 MHz, Unit: Hz

set_gauss(*rbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:GAUSs
driver.configure.gprfMeasurement.spectrum.zeroSpan.rbw.set_gauss(rbw = 1.0)
```

Selects the bandwidth of the Gaussian resolution filter for the zero span mode. To use this filter, configure also method RsCma.Configure.GprfMeasurement.Spectrum.ZeroSpan.Rbw.typePy.

param rbw
You can enter values between 100 Hz and 10 MHz. The setting is rounded to the closest of the following values: 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz Unit: Hz

set_type_py(*rbw_type: RbwFilterType*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:RBW:TYPE
driver.configure.gprfMeasurement.spectrum.zeroSpan.rbw.set_type_py(rbw_type =␣
↪enums.RbwFilterType.BANDpass)
```

Selects the resolution filter type for the zero span mode.

param rbw_type
GAUSs | BANDpass GAUSs Gaussian filter with configurable bandwidth BANDpass Bandpass filter with 40 MHz bandwidth

6.3.4.8.6.5 Vbw

SCPI Command:

```
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.gprfMeasurement.spectrum.zeroSpan.vbw.get_auto()
```

Enables or disables automatic configuration of the video bandwidth (VBW) for the 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.gprfMeasurement.spectrum.zeroSpan.vbw.
↪get_value()
```

Selects the bandwidth of the video filter for the zero span mode. Setting this value is only possible if the automatic VBW selection is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.ZeroSpan.Vbw.auto.

return
vbw: (float or boolean) You can enter values between 10 Hz and 10 MHz. The setting is rounded to the closest of the following values: 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz OFF disables the filter Unit: Hz

set_auto(*vbw_auto: bool*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW:AUTO
driver.configure.gprfMeasurement.spectrum.zeroSpan.vbw.set_auto(vbw_auto =␣
↪False)
```

Enables or disables automatic configuration of the video bandwidth (VBW) for the zero span mode.

param vbw_auto
OFF | ON

set_value(*vbw: float*) → None

```
# SCPI: CONFigure:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:VBW
driver.configure.gprfMeasurement.spectrum.zeroSpan.vbw.set_value(vbw = 1.0)
```

Selects the bandwidth of the video filter for the zero span mode. Setting this value is only possible if the automatic VBW selection is switched off, see method RsCma.Configure.GprfMeasurement.Spectrum.ZeroSpan.Vbw.auto.

param vbw
(float or boolean) You can enter values between 10 Hz and 10 MHz. The setting is rounded to the closest of the following values: 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 Hz 1 / 2 / 3 / 5 / 10 / 20 / 30 / 50 / 100 / 200 / 300 / 500 kHz 1 / 2 / 3 / 5 / 10 MHz OFF disables the filter Unit: Hz

6.3.5 Sequencer

class SequencerCls

Sequencer 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.sequencer.clone()
```

Subgroups

6.3.5.1 Tplan

SCPI Command:

```
CONFigure:SEQuencer:TPLan:RUN
CONFigure:SEQuencer:TPLan:ABORt
```

class TplanCls

Tplan commands group definition. 2 total commands, 0 Subgroups, 2 group commands

abort(*test_plan_name: str*) → None

```
# SCPI: CONFigure:SEQuencer:TPLan:ABORt
driver.configure.sequencer.tplan.abort(test_plan_name = '1')
```

No command help available

param test_plan_name
No help available

run(*test_plan_name: str*) → None

```
# SCPI: CONFigure:SEQuencer:TPLan:RUN
driver.configure.sequencer.tplan.run(test_plan_name = '1')
```

No command help available

param test_plan_name
No help available

6.3.6 Vse

class VseCls

Vse commands group definition. 67 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.clone()
```

Subgroups

6.3.6.1 Measurement

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:CREPetition
CONFigure:VSE:MEASurement<Instance>:SCOunt
CONFigure:VSE:MEASurement<Instance>:REPetition
CONFigure:VSE:MEASurement<Instance>:SCONdition
CONFigure:VSE:MEASurement<Instance>:RCOupling
CONFigure:VSE:MEASurement<Instance>:TOUT
CONFigure:VSE:MEASurement<Instance>:STANdard
```

class MeasurementCls

Measurement commands group definition. 67 total commands, 10 Subgroups, 7 group commands

get_crepetition() → bool

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:CREPetition
value: bool = driver.configure.vse.measurement.get_crepetition()
```

Enables or disables the automatic configuration of the repetition mode. With enabled automatic configuration, the repetition mode of all measurements is set to 'Continuous' each time the instrument switches from remote operation to manual operation.

return
continuous_repetition: No help available

get_rcoupling() → bool

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RCOupling
value: bool = driver.configure.vse.measurement.get_rcoupling()
```

Couples the repetition mode (single shot or continuous) of all measurements.

return
repetition_coupling: OFF | ON

get_repetition() → Repeat

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:REPetition
value: enums.Repeat = driver.configure.vse.measurement.get_repetition()
```

Selects whether the measurement is repeated continuously or not.

return
repetition: SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after the statistic count CONTinuous Continuous measurement, running until explicitly terminated

get_scondition() → StopCondition

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:SCONdition
value: enums.StopCondition = driver.configure.vse.measurement.get_scondition()
```

No command help available

return
stop_condition: No help available

get_scount() → int

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:SCOunt
value: int = driver.configure.vse.measurement.get_scount()
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval delivers a single 'Current' value per result.

return
statistic_count: Range: 1 to 1000

get_standard() → Standard

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:STANdard
value: enums.Standard = driver.configure.vse.measurement.get_standard()
```

Selects the digital standard of the measured signal.

return
standard: DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom

get_timeout() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:TOUT
value: float = driver.configure.vse.measurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
tcd_timeout: No help available

set_crepetition(*continuous_repetition: bool*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:CREPetition
driver.configure.vse.measurement.set_crepetition(continuous_repetition = False)
```

Enables or disables the automatic configuration of the repetition mode. With enabled automatic configuration, the repetition mode of all measurements is set to 'Continuous' each time the instrument switches from remote operation to manual operation.

param continuous_repetition
OFF | ON

set_rcoupling(*repetition_coupling: bool*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RCOupling
driver.configure.vse.measurement.set_rcoupling(repetition_coupling = False)
```

Couples the repetition mode (single shot or continuous) of all measurements.

param repetition_coupling
OFF | ON

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:REPetition
driver.configure.vse.measurement.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Selects whether the measurement is repeated continuously or not.

param repetition
SINGleshot | CONTinuous SINGleshot Single-shot measurement, stopped after the statistic count CONTinuous Continuous measurement, running until explicitly terminated

set_scondition(*stop_condition: StopCondition*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:SCONdition
driver.configure.vse.measurement.set_scondition(stop_condition = enums.
↪StopCondition.NONE)
```

No command help available

param stop_condition
No help available

set_scount(*statistic_count: int*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:SCOunt
driver.configure.vse.measurement.set_scount(statistic_count = 1)
```

Specifies the number of measurement intervals per measurement cycle. One measurement interval delivers a single 'Current' value per result.

param statistic_count
Range: 1 to 1000

set_standard(*standard: Standard*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:STANdard
driver.configure.vse.measurement.set_standard(standard = enums.Standard.CUSTom)
```

Selects the digital standard of the measured signal.

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom

set_timeout(*tcd_timeout: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:TOUT
driver.configure.vse.measurement.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 via the graphical user interface. The timer is reset after the first measurement cycle. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped and the reliability indicator is set to 1. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results,

there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param tcd_timeout
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.clone()
```

Subgroups

6.3.6.1.1 Custom

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:CUSTom:LOAD
CONFigure:VSE:MEASurement<Instance>:CUSTom:SAVE
```

class CustomCls

Custom commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_load() → str

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:CUSTom:LOAD
value: str = driver.configure.vse.measurement.custom.get_load()
```

Loads a VSE recall file with file extension *.dfl. You can load VSE recall files from the data directory of the internal storage medium (D:/Rohde-Schwarz/CMA/Data or subfolder) or from a USB storage medium.

return
filename: No help available

save(*filename: str*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:CUSTom:SAVE
driver.configure.vse.measurement.custom.save(filename = '1')
```

Saves a VSE recall file with file extension *.dfl. You can save VSE recall files to the data directory of the internal storage medium (D:/Rohde-Schwarz/CMA/Data or subfolder) or to a USB storage medium.

param filename
No help available

set_load(*filename: str*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:CUSTom:LOAD
driver.configure.vse.measurement.custom.set_load(filename = '1')
```

Loads a VSE recall file with file extension *.dfl. You can load VSE recall files from the data directory of the internal storage medium (D:/Rohde-Schwarz/CMA/Data or subfolder) or from a USB storage medium.

param filename
No help available

6.3.6.1.2 Dmr

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:DMR:SRATe
CONFigure:VSE:MEASurement<Instance>:DMR:DEModulation
```

class DmrCls

Dmr commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_demodulation() → DemodulationType

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:DMR:DEModulation
value: enums.DemodulationType = driver.configure.vse.measurement.dmr.get_
↪demodulation()
```

Queries the modulation type used for DMR.

return
demodulation_type: FSK4

get_symbol_rate() → int

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:DMR:SRATe
value: int = driver.configure.vse.measurement.dmr.get_symbol_rate()
```

Queries the symbol rate for DMR.

return
symbol_rate: Range: 4800 symbol/s to 4800 symbol/s , Unit: symbol/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.dmr.clone()
```

Subgroups

6.3.6.1.2.1 FilterPy

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:DMR:FILTer
```

class FilterPyCls

FilterPy commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → PulseShapingUserFilter

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:DMR:FILTer
value: enums.PulseShapingUserFilter = driver.configure.vse.measurement.dmr.
↪filterPy.get_value()
```

Queries the filter type used for pulse shaping for DMR.

return
filter_py: RCOS

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.dmr.filterPy.clone()
```

Subgroups

6.3.6.1.2.2 Rrc

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:DMR:FILTer:RRC:ROFFfactor
```

class RrcCls

Rrc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_roff_factor() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:DMR:FILTer:RRC:ROFFfactor
value: float = driver.configure.vse.measurement.dmr.filterPy.rrc.get_roff_
↪factor()
```

Queries the roll-off factor of the filter used for pulse shaping for DMR.

return
rolloff_factor: Range: 0.2 to 0.2

6.3.6.1.3 Dpmr

class DpmrCls

Dpmr 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.vse.measurement.dpmr.clone()
```

Subgroups

6.3.6.1.3.1 FilterPy

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:DPMR:FILTer
```

class FilterPyCls

FilterPy commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → PulseShapingUserFilter

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:DPMR:FILTer
value: enums.PulseShapingUserFilter = driver.configure.vse.measurement.dpmr.
↪filterPy.get_value()
```

Queries the filter type used for pulse shaping for DPMR.

return
filter_py: RRC

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.dpmr.filterPy.clone()
```

Subgroups

6.3.6.1.3.2 Rrc

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:DPMR:FILTer:RRC:ROFFfactor
```

class RrcCls

Rrc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_roff_factor() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:DPMR:FILTer:RRC:ROFFfactor
value: float = driver.configure.vse.measurement.dpmr.filterPy.rrc.get_roff_
↪factor()
```

Queries the roll-off factor of the filter used for pulse shaping for DPMR.

return
rolloff_factor: Range: 0.2 to 0.2

6.3.6.1.4 IqRecorder

class IqRecorderCls

IqRecorder commands group definition. 8 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.iqRecorder.clone()
```

Subgroups

6.3.6.1.4.1 Capture

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:CAPTure
```

class CaptureCls

Capture commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Capt_Pre_Trig: int: Range: 1 Samples to 4194303
- Capt_Post_Trig: int: Range: 1 Samples to 4194303

get() → GetStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:CAPTure
value: GetStruct = driver.configure.vse.measurement.iqRecorder.capture.get()
```

Sets or queries the number of samples to be received before IQ recording is started (‘Pre Trigger’) and after which IQ recording is stopped (‘Post Trigger’) .

return
structure: for return value, see the help for GetStruct structure arguments.

set(*capt_pre_trig: int*, *capt_post_trig: int = None*, *standard: Standard = None*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:CAPTure
driver.configure.vse.measurement.iqRecorder.capture.set(capt_pre_trig = 1, capt_
↪post_trig = 1, standard = enums.Standard.CUSTom)
```

Sets or queries the number of samples to be received before IQ recording is started (‘Pre Trigger’) and after which IQ recording is stopped (‘Post Trigger’) .

param capt_pre_trig
Range: 1 Samples to 4194303

param capt_post_trig
Range: 1 Samples to 4194303

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom DMR | DPMR | NXDN | P25 | TETRa | LTE Allows query of the number of samples. CUSTom Allows set and query of the number of samples.

6.3.6.1.4.2 FilterPy

class FilterPyCls

FilterPy commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.iqRecorder.filterPy.clone()
```

Subgroups

6.3.6.1.4.3 Bandpass

class BandpassCls

Bandpass commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.iqRecorder.filterPy.bandpass.clone()
```

Subgroups

6.3.6.1.4.4 Bandwidth

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:BANDpass:BWIDth
```

class BandwidthCls

Bandwidth commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:BANDpass:BWIDth
value: float = driver.configure.vse.measurement.iqRecorder.filterPy.bandpass.
↪bandwidth.get()
```

Sets or queries the bandwidth of the bandpass filter and the related digital standard.

return
bandpass_bw: Range: 1000 Hz to 160 MHz, Unit: Hz

set(*bandpass_bw: float*, *standard: Standard = None*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:BANDpass:BWIDth
driver.configure.vse.measurement.iqRecorder.filterPy.bandpass.bandwidth.
↪set(bandpass_bw = 1.0, standard = enums.Standard.CUSTom)
```

Sets or queries the bandwidth of the bandpass filter and the related digital standard.

param bandpass_bw
Range: 1000 Hz to 160 MHz, Unit: Hz

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom DMR | DPMR | NXDN | P25 | TETRa | LTE Allows query of the bandwidth of the bandpass filter. CUSTom Allows set and query of the bandwidth of the bandpass filter.

6.3.6.1.4.5 Gauss

class GaussCls

Gauss commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.iqRecorder.filterPy.gauss.clone()
```

Subgroups

6.3.6.1.4.6 Bandwidth

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:GAUSs:BWIDth
```

class BandwidthCls

Bandwidth commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:GAUSs:BWIDth
value: float = driver.configure.vse.measurement.iqRecorder.filterPy.gauss.
↪bandwidth.get()
```

Sets or queries the bandwidth of the Gauss filter and the related digital standard.

return
gauss_bw: Range: 1000 Hz to 10 MHz, Unit: Hz

set(*gauss_bw: float*, *standard: Standard = None*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:GAUSs:BWIDth
driver.configure.vse.measurement.iqRecorder.filterPy.gauss.bandwidth.set(gauss_
↪bw = 1.0, standard = enums.Standard.CUSTom)
```

Sets or queries the bandwidth of the Gauss filter and the related digital standard.

param gauss_bw
Range: 1000 Hz to 10 MHz, Unit: Hz

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom DMR | DPMR | NXDN | P25 | TETRa | LTE Allows query of the bandwidth of the Gauss filter. CUSTom Allows set and query of the bandwidth of the Gauss filter. Unit: Hz

6.3.6.1.4.7 TypePy

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:TYPE
```

class TypePyCls

TypePy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → RbwFilterType

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:TYPE
value: enums.RbwFilterType = driver.configure.vse.measurement.iqRecorder.
↪filterPy.typePy.get()
```

Selects or queries the type of filter to be applied to the data after demodulation and the related digital standard.

return
filter_type: BANDpass | GAUSs

set(*filter_type: RbwFilterType*, *standard: Standard = None*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:FILTer:TYPE
driver.configure.vse.measurement.iqRecorder.filterPy.typePy.set(filter_type =␣
↪enums.RbwFilterType.BANDpass, standard = enums.Standard.CUSTom)
```

Selects or queries the type of filter to be applied to the data after demodulation and the related digital standard.

param filter_type
BANDpass | GAUSs

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom DMR | DPMR | NXDN | P25 | TETRa | LTE Allows query of the filter type, that is a bandpass filter. CUSTom Allows set and query of the filter type.

6.3.6.1.4.8 Lte

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:LTE:CBWidth
```

class LteCls

Lte commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_cbwidth() → LteChannelBandwidth

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:LTE:CBWidth
value: enums.LteChannelBandwidth = driver.configure.vse.measurement.iqRecorder.
↪lte.get_cbwidth()
```

Sets the 'LTE' channel bandwidth of the received RF signals for I/Q data analysis.

return
channel_bandwidth: F3M | F5M | F10M | F20M

set_cbwidth(*channel_bandwidth: LteChannelBandwidth*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:LTE:CBWidth
driver.configure.vse.measurement.iqRecorder.lte.set_cbwidth(channel_bandwidth =␣
↪enums.LteChannelBandwidth.F10M)
```

Sets the 'LTE' channel bandwidth of the received RF signals for I/Q data analysis.

param channel_bandwidth
F3M | F5M | F10M | F20M

6.3.6.1.4.9 Munit

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:MUNit
```

class MunitCls

Munit commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → MagnitudeUnit

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:MUNit
value: enums.MagnitudeUnit = driver.configure.vse.measurement.iqRecorder.munit.
↪get()
```

Selects or queries the physical unit for representation of the recorded I/Q data.

return
magnitude_unit: VOLT | RAW

set(*magnitude_unit: MagnitudeUnit*, *standard: Standard = None*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:MUNit
driver.configure.vse.measurement.iqRecorder.munit.set(magnitude_unit = enums.
↪MagnitudeUnit.RAW, standard = enums.Standard.CUSTom)
```

Selects or queries the physical unit for representation of the recorded I/Q data.

param magnitude_unit
VOLT | RAW

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom DMR | DPMR | NXDN | P25 | TETRa | LTE Allows query of the physical unit. CUSTom Allows set and query of the physical unit.

6.3.6.1.4.10 Ratio

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:RATio
```

class RatioCls

Ratio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:RATio
value: float = driver.configure.vse.measurement.iqRecorder.ratio.get()
```

Sets or queries the ratio of ‘Sample Rate’ and ‘Max. Sample Rate’.

return
ratio: Range: 1E-3 to 1

set(*ratio: float*, *standard: Standard = None*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:RATio
driver.configure.vse.measurement.iqRecorder.ratio.set(ratio = 1.0, standard =␣
↪enums.Standard.CUSTom)
```

Sets or queries the ratio of ‘Sample Rate’ and ‘Max. Sample Rate’.

param ratio
Range: 1E-3 to 1

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom DMR | DPMR | NXDN | P25 | TETRa | LTE Allows query of the ratio of ‘Sample Rate’ and ‘Max. Sample Rate’. CUSTom Allows set and query of the ratio of ‘Sample Rate’ and ‘Max. Sample Rate’.

6.3.6.1.4.11 SymbolRate

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:IQRecorder:SRATe
```

class SymbolRateCls

SymbolRate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:SRATe
value: float = driver.configure.vse.measurement.iqRecorder.symbolRate.get()
```

Sets or queries the sample rate of the IQ recorder.

return
sample_rate: Range: 0 Hz to 200 MHz, Unit: Hz

set(*sample_rate: float*, *standard: Standard = None*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:IQRecorder:SRATe
driver.configure.vse.measurement.iqRecorder.symbolRate.set(sample_rate = 1.0,␣
↪standard = enums.Standard.CUSTom)
```

Sets or queries the sample rate of the IQ recorder.

param sample_rate
Range: 0 Hz to 200 MHz, Unit: Hz

param standard
DMR | DPMR | NXDN | P25 | TETRa | LTE | SPECtrum | CUSTom DMR | DPMR | NXDN | P25 | TETRa | LTE Allows query of the sample rate of the IQ recorder. CUSTom Allows set and query of the sample rate of the IQ recorder.

6.3.6.1.5 Limit

class LimitCls

Limit commands group definition. 26 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.clone()
```

Subgroups

6.3.6.1.5.1 Dmr

class DmrCls

Dmr commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.dmr.clone()
```

Subgroups

6.3.6.1.5.2 FdError

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FDERor
```

class FdErrorCls

FdError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FdErrorStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper FSK deviation error limit Range: 0 Hz to 1 MHz, Unit: Hz

get() → FdErrorStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FDERor
value: FdErrorStruct = driver.configure.vse.measurement.limit.dmr.fdError.get()
```

Configures an upper limit for the measured FSK deviation error for the digital standard 'DMR'.

return
structure: for return value, see the help for FdErrorStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FDERor
driver.configure.vse.measurement.limit.dmr.fdError.set(enable = False, limit =␣
↪1.0)
```

Configures an upper limit for the measured FSK deviation error for the digital standard 'DMR'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper FSK deviation error limit Range: 0 Hz to 1 MHz, Unit: Hz

6.3.6.1.5.3 FfError

class FfErrorCls

FfError commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.dmr.ffError.clone()
```

Subgroups

6.3.6.1.5.4 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FFERor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper FSK frequency error limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FFERor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.dmr.ffError.peak.
↪get()
```

Configures an upper limit for the measured peak value of the FSK frequency error for the digital standard ‘DMR’.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FFERor:PEAK
driver.configure.vse.measurement.limit.dmr.ffError.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the FSK frequency error for the digital standard ‘DMR’.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper FSK frequency error limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.5 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FFERor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper FSK frequency error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FFERor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.dmr.ffError.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the FSK frequency error for the digital standard 'DMR'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:FFERor:RMS
driver.configure.vse.measurement.limit.dmr.ffError.rms.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured 'RMS' value of the FSK frequency error for the digital standard 'DMR'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper FSK frequency error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.6 Merror

class MerrorCls

Merror commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.dmr.merror.clone()
```

Subgroups

6.3.6.1.5.7 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:MERRor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:MERRor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.dmr.merror.peak.get()
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard 'DMR'.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:MERRor:PEAK
driver.configure.vse.measurement.limit.dmr.merror.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard 'DMR'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.8 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:MERRor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:MERRor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.dmr.merror.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'DMR'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DMR:MERRor:RMS
driver.configure.vse.measurement.limit.dmr.merror.rms.set(enable = False, limit␣
↪= 1.0)
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'DMR'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.9 Dpmr

class DpmrCls

Dpmr commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.dpmr.clone()
```

Subgroups

6.3.6.1.5.10 FdError

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FDERor
```

class FdErrorCls

FdError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FdErrorStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: Upper FSK deviation error limit Range: 0 Hz to 1 MHz, Unit: Hz

get() → FdErrorStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FDERor
value: FdErrorStruct = driver.configure.vse.measurement.limit.dpmr.fdError.get()
```

Configures an upper limit for the measured FSK deviation error for the digital standard 'DPMR'.

return
structure: for return value, see the help for FdErrorStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FDERor
driver.configure.vse.measurement.limit.dpmr.fdError.set(enable = False, limit =␣
↪1.0)
```

Configures an upper limit for the measured FSK deviation error for the digital standard 'DPMR'.

param enable
OFF | ON

param limit
Upper FSK deviation error limit Range: 0 Hz to 1 MHz, Unit: Hz

6.3.6.1.5.11 FfError

class FfErrorCls

FfError commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.dpmr.ffError.clone()
```

Subgroups

6.3.6.1.5.12 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FFERor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper FSK frequency error limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FFERor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.dpmr.ffError.peak.
↪get()
```

Configures an upper limit for the measured peak value of the FSK frequency error for the digital standard ‘DPMR’.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FFERor:PEAK
driver.configure.vse.measurement.limit.dpmr.ffError.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the FSK frequency error for the digital standard ‘DPMR’.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper FSK frequency error limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.13 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FFERor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: Upper FSK frequency error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FFERor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.dpmr.ffError.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the FSK frequency error for the digital standard 'DPMR'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:FFERor:RMS
driver.configure.vse.measurement.limit.dpmr.ffError.rms.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured 'RMS' value of the FSK frequency error for the digital standard 'DPMR'.

param enable
OFF | ON

param limit
Upper FSK frequency error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.14 Merror

class MerrorCls

Merror commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.dpmr.merror.clone()
```

Subgroups

6.3.6.1.5.15 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:MERRor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:MERRor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.dpmr.merror.peak.
↪get()
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard ‘DPMR’.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:MERRor:PEAK
driver.configure.vse.measurement.limit.dpmr.merror.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard ‘DPMR’.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.16 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:MERRor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:MERRor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.dpmr.merror.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'DPMR'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:DPMR:MERRor:RMS
driver.configure.vse.measurement.limit.dpmr.merror.rms.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'DPMR'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.17 Nxdn

class NxdnCls

Nxdn commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.nxdn.clone()
```

Subgroups

6.3.6.1.5.18 FdError

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FDERor
```

class FdErrorCls

FdError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FdErrorStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper FSK deviation error limit Range: 0 Hz to 1 MHz, Unit: Hz

get() → FdErrorStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FDERor
value: FdErrorStruct = driver.configure.vse.measurement.limit.nxdn.fdError.get()
```

Configures an upper limit for the measured FSK deviation error for the digital standard 'NXDN'.

return
structure: for return value, see the help for FdErrorStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FDERor
driver.configure.vse.measurement.limit.nxdn.fdError.set(enable = False, limit =␣
↪1.0)
```

Configures an upper limit for the measured FSK deviation error for the digital standard 'NXDN'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper FSK deviation error limit Range: 0 Hz to 1 MHz, Unit: Hz

6.3.6.1.5.19 FfError

class FfErrorCls

FfError commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.nxdn.ffError.clone()
```

Subgroups

6.3.6.1.5.20 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FFERor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: Upper FSK frequency error limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FFERor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.nxdn.ffError.peak.
↪get()
```

Configures an upper limit for the measured peak value of the FSK frequency error for the digital standard ‘NXDN’.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FFERor:PEAK
driver.configure.vse.measurement.limit.nxdn.ffError.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the FSK frequency error for the digital standard ‘NXDN’.

param enable
OFF | ON

param limit
Upper FSK frequency error limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.21 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FFERor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper FSK frequency error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FFERor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.nxdn.ffError.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the FSK frequency error for the digital standard 'NXDN'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:FFERor:RMS
driver.configure.vse.measurement.limit.nxdn.ffError.rms.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured 'RMS' value of the FSK frequency error for the digital standard 'NXDN'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper FSK frequency error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.22 Merror

class MerrorCls

Merror commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.nxdn.merror.clone()
```

Subgroups

6.3.6.1.5.23 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:MERRor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:MERRor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.nxdn.merror.peak.
↪get()
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard 'NXDN'.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:MERRor:PEAK
driver.configure.vse.measurement.limit.nxdn.merror.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard 'NXDN'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.24 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:MERRor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:MERRor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.nxdn.merror.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'NXDN'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:NXDN:MERRor:RMS
driver.configure.vse.measurement.limit.nxdn.merror.rms.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'NXDN'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.25 PtFive

class PtFiveCls

PtFive commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.ptFive.clone()
```

Subgroups

6.3.6.1.5.26 FdError

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FDERor
```

class FdErrorCls

FdError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FdErrorStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: Range: 0 Hz to 1 MHz , Unit: Hz

get() → FdErrorStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FDERor
value: FdErrorStruct = driver.configure.vse.measurement.limit.ptFive.fdError.
↪get()
```

Enables/disables limit evaluation and sets the upper limit for the FSK deviation error.

return
structure: for return value, see the help for FdErrorStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FDERor
driver.configure.vse.measurement.limit.ptFive.fdError.set(enable = False, limit␣
↪= 1.0)
```

Enables/disables limit evaluation and sets the upper limit for the FSK deviation error.

param enable
OFF | ON

param limit
Range: 0 Hz to 1 MHz , Unit: Hz

6.3.6.1.5.27 FfError

class FfErrorCls

FfError commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.ptFive.ffError.clone()
```

Subgroups

6.3.6.1.5.28 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FFERor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: 0.00 % to 100.00 % Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FFERor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.ptFive.ffError.peak.
↪get()
```

Configures an upper limit for the measured peak value of the frequency deviation error.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FFERor:PEAK
driver.configure.vse.measurement.limit.ptFive.ffError.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the frequency deviation error.

param enable
OFF | ON

param limit
0.00 % to 100.00 % Unit: %

6.3.6.1.5.29 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FFERor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: Range: 0.00 % to 100.00 % , Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FFERor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.ptFive.ffError.rms.
↪get()
```

Configures an upper limit for the measured RMS value of the frequency deviation error.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:FFERor:RMS
driver.configure.vse.measurement.limit.ptFive.ffError.rms.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured RMS value of the frequency deviation error.

param enable
OFF | ON

param limit
Range: 0.00 % to 100.00 % , Unit: %

6.3.6.1.5.30 Mfidelity

class MfidelityCls

Mfidelity commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.ptFive.mfidelity.clone()
```

Subgroups

6.3.6.1.5.31 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:MFIDelity:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: Range: 0.00 % to 100.00 % , Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:MFIDelity:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.ptFive.mfidelity.
↪peak.get()
```

Enables/disables limit evaluation and sets the upper limit for the peak modulation fidelity.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:MFIDelity:PEAK
driver.configure.vse.measurement.limit.ptFive.mfidelity.peak.set(enable = False,
↪ limit = 1.0)
```

Enables/disables limit evaluation and sets the upper limit for the peak modulation fidelity.

param enable
OFF | ON

param limit
Range: 0.00 % to 100.00 % , Unit: %

6.3.6.1.5.32 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:MFIDelity:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Limit: float: Range: 0.00 % to 100.00 % , Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:MFIDelity:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.ptFive.mfidelity.rms.
↪get()
```

Enables/disables limit evaluation and sets the upper limit for the RMS modulation fidelity.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:PTFive:MFIDelity:RMS
driver.configure.vse.measurement.limit.ptFive.mfidelity.rms.set(enable = False,␣
↪limit = 1.0)
```

Enables/disables limit evaluation and sets the upper limit for the RMS modulation fidelity.

param enable
OFF | ON

param limit
Range: 0.00 % to 100.00 % , Unit: %

6.3.6.1.5.33 RfCarrier

class RfCarrierCls

RfCarrier commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.rfCarrier.clone()
```

Subgroups

6.3.6.1.5.34 FreqError

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:RFCarrier:FERRor
```

class FreqErrorCls

FreqError commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FreqErrorStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Range: 0 Hz to 1 MHz, Unit: Hz

get() → FreqErrorStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:RFCarrier:FERRor
value: FreqErrorStruct = driver.configure.vse.measurement.limit.rfCarrier.
↪freqError.get()
```

Configures an upper limit for the measured RF carrier frequency error.

return
structure: for return value, see the help for FreqErrorStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:RFCarrier:FERRor
driver.configure.vse.measurement.limit.rfCarrier.freqError.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured RF carrier frequency error.

param enable
OFF | ON Enables or disables the limit check

param limit
Range: 0 Hz to 1 MHz, Unit: Hz

6.3.6.1.5.35 Power

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:RFCarrier:POWer
```

class PowerCls

Power commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PowerStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Lower: float: Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm
- Upper: float: Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

get() → PowerStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:RFCarrier:POWer
value: PowerStruct = driver.configure.vse.measurement.limit.rfCarrier.power.
↪get()
```

Configures limits for the measured RF power (PEP) .

return
structure: for return value, see the help for PowerStruct structure arguments.

set(*enable: bool*, *lower: float*, *upper: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:RFCarrier:POWer
driver.configure.vse.measurement.limit.rfCarrier.power.set(enable = False,␣
↪lower = 1.0, upper = 1.0)
```

Configures limits for the measured RF power (PEP) .

param enable
OFF | ON Enables or disables the limit check

param lower
Lower power limit Range: -130 dBm to 55 dBm, Unit: dBm

param upper
Upper power limit Range: -130 dBm to 55 dBm, Unit: dBm

6.3.6.1.5.36 Tetra

class TetraCls

Tetra commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.tetra.clone()
```

Subgroups

6.3.6.1.5.37 Evm

class EvmCls

Evm commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.tetra.evm.clone()
```

Subgroups

6.3.6.1.5.38 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:EVM:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper EVM limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:EVM:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.tetra.evm.peak.get()
```

Configures an upper limit for the measured peak value of the EVM value for the digital standard 'TETRA'.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:EVM:PEAK
driver.configure.vse.measurement.limit.tetra.evm.peak.set(enable = False, limit␣
↪= 1.0)
```

Configures an upper limit for the measured peak value of the EVM value for the digital standard 'TETRA'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper EVM limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.39 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:EVM:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper EVM limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:EVM:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.tetra.evm.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the EVM value for the digital standard 'TETRA'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:EVM:RMS
driver.configure.vse.measurement.limit.tetra.evm.rms.set(enable = False, limit␣
↪= 1.0)
```

Configures an upper limit for the measured 'RMS' value of the EVM value for the digital standard 'TETRA'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper EVM limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.40 Merror

class MerrorCls

Merror commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.limit.tetra.merror.clone()
```

Subgroups

6.3.6.1.5.41 Peak

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:MERRor:PEAK
```

class PeakCls

Peak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class PeakStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

get() → PeakStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:MERRor:PEAK
value: PeakStruct = driver.configure.vse.measurement.limit.tetra.merror.peak.
↪get()
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard 'TETRA'.

return
structure: for return value, see the help for PeakStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:MERRor:PEAK
driver.configure.vse.measurement.limit.tetra.merror.peak.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured peak value of the magnitude error for the digital standard 'TETRA'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for peak value Range: 0 % to 100 %, Unit: %

6.3.6.1.5.42 Rms

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:MERRor:RMS
```

class RmsCls

Rms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RmsStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the limit check
- Limit: float: Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

get() → RmsStruct

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:MERRor:RMS
value: RmsStruct = driver.configure.vse.measurement.limit.tetra.merror.rms.get()
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'TETRA'.

return
structure: for return value, see the help for RmsStruct structure arguments.

set(*enable: bool*, *limit: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:LIMit:TETRa:MERRor:RMS
driver.configure.vse.measurement.limit.tetra.merror.rms.set(enable = False,␣
↪limit = 1.0)
```

Configures an upper limit for the measured 'RMS' value of the magnitude error for the digital standard 'TETRA'.

param enable
OFF | ON Enables or disables the limit check

param limit
Upper magnitude error limit for 'RMS' value Range: 0 % to 100 %, Unit: %

6.3.6.1.6 Nxdn

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:NXDN:TRANsmission
```

class NxdnCls

Nxdn commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_transmission() → Transmission

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:NXDN:TRANsmission
value: enums.Transmission = driver.configure.vse.measurement.nxdn.get_
↪transmission()
```

Queries the data rate in bits/s for enhanced full rate (EFR) or enhanced half rate (EHR) .

return
transmission: EHR4800 | EHR9600 | EFR9600

set_transmission(*transmission: Transmission*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:NXDN:TRANsmission
driver.configure.vse.measurement.nxdn.set_transmission(transmission = enums.
↪Transmission.EFR9600)
```

Queries the data rate in bits/s for enhanced full rate (EFR) or enhanced half rate (EHR) .

param transmission
EHR4800 | EHR9600 | EFR9600

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.nxdn.clone()
```

Subgroups

6.3.6.1.6.1 FilterPy

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:NXDN:FILTer
```

class FilterPyCls

FilterPy commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → PulseShapingUserFilter

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:NXDN:FILTer
value: enums.PulseShapingUserFilter = driver.configure.vse.measurement.nxdn.
↪filterPy.get_value()
```

No command help available

return
filter_py: GAUSs | RRC | COS | SINC | NXRX

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.nxdn.filterPy.clone()
```

Subgroups

6.3.6.1.6.2 Rrc

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:NXDN:FILTer:RRC:ROFFfactor
```

class RrcCls

Rrc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_roff_factor() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:NXDN:FILTer:RRC:ROFFfactor
value: float = driver.configure.vse.measurement.nxdn.filterPy.rrc.get_roff_
↪factor()
```

Queries the roll-off factor of the filter used for pulse shaping for NXDN.

return
rolloff_factor: No help available

6.3.6.1.7 PtFive

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:PTFive:SRATe
CONFigure:VSE:MEASurement<Instance>:PTFive:MODE
CONFigure:VSE:MEASurement<Instance>:PTFive:FILTer
```

class PtFiveCls

PtFive commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_filter_py() → PulseShapingUserFilter

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:PTFive:FILTer
value: enums.PulseShapingUserFilter = driver.configure.vse.measurement.ptFive.
↪get_filter_py()
```

Specifies the used P25 measurement filter.

return
filter_py: SINC

get_mode() → P25Mode

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:PTFive:MODE
value: enums.P25Mode = driver.configure.vse.measurement.ptFive.get_mode()
```

Specifies the modulation type used for P25 phase 1 modulation.

return
mode: C4FM

get_symbol_rate() → int

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:PTFive:SRATe
value: int = driver.configure.vse.measurement.ptFive.get_symbol_rate()
```

Queries the symbol rate for P25 with C4FM modulation.

return
symbol_rate: Unit: symbol/s

6.3.6.1.8 Result

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:RESult:EDIagram
CONFigure:VSE:MEASurement<Instance>:RESult:PVTime
CONFigure:VSE:MEASurement<Instance>:RESult:CONS
CONFigure:VSE:MEASurement<Instance>:RESult:SDIS
```

class ResultCls

Result commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_cons() → bool

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:CONS
value: bool = driver.configure.vse.measurement.result.get_cons()
```

Enables or disables the measurement of the constellation diagram.

return
constellation_enable: OFF | ON

get_ediagram() → bool

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:EDIagram
value: bool = driver.configure.vse.measurement.result.get_ediagram()
```

Enables or disables the measurement of the eye diagram.

return
eye_diagram_enable: OFF | ON

get_power_vs_time() → bool

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:PVTime
value: bool = driver.configure.vse.measurement.result.get_power_vs_time()
```

Enables or disables the measurement of the power vs. time results.

return
power_vs_time_enable: OFF | ON

get_sdis() → bool

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:SDIS
value: bool = driver.configure.vse.measurement.result.get_sdis()
```

Enables or disables the measurement of the symbol distribution results.

return
sdistribution_enable: No help available

set_cons(*constellation_enable: bool*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:CONS
driver.configure.vse.measurement.result.set_cons(constellation_enable = False)
```

Enables or disables the measurement of the constellation diagram.

param constellation_enable
OFF | ON

set_ediagram(*eye_diagram_enable: bool*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:EDIagram
driver.configure.vse.measurement.result.set_ediagram(eye_diagram_enable = False)
```

Enables or disables the measurement of the eye diagram.

param eye_diagram_enable
OFF | ON

set_power_vs_time(*power_vs_time_enable: bool*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:PVTime
driver.configure.vse.measurement.result.set_power_vs_time(power_vs_time_enable␣
↪= False)
```

Enables or disables the measurement of the power vs. time results.

param power_vs_time_enable
OFF | ON

set_sdis(*sdistribution_enable: bool*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:RESult:SDIS
driver.configure.vse.measurement.result.set_sdis(sdistribution_enable = False)
```

Enables or disables the measurement of the symbol distribution results.

param sdistribution_enable
OFF | ON

6.3.6.1.9 Tetra

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:TETRa:SRATe
CONFigure:VSE:MEASurement<Instance>:TETRa:DEModulation
```

class TetraCls

Tetra commands group definition. 4 total commands, 1 Subgroups, 2 group commands

get_demodulation() → DemodulationType

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:TETRa:DEModulation
value: enums.DemodulationType = driver.configure.vse.measurement.tetra.get_
↪demodulation()
```

Queries the modulation type used for TETRA.

return
demodulation_type: FSK4

get_symbol_rate() → int

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:TETRa:SRATe
value: int = driver.configure.vse.measurement.tetra.get_symbol_rate()
```

Queries the symbol rate for TETRA.

return
symbol_rate: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.tetra.clone()
```

Subgroups

6.3.6.1.9.1 FilterPy

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:TETRa:FILTer
```

class FilterPyCls

FilterPy commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → PulseShapingUserFilter

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:TETRa:FILTer
value: enums.PulseShapingUserFilter = driver.configure.vse.measurement.tetra.
↪filterPy.get_value()
```

Queries the filter type used for pulse shaping for TETRA.

return
filter_py: RRC

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.tetra.filterPy.clone()
```

Subgroups

6.3.6.1.9.2 Rrc

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:TETRa:FILTer:RRC:ROFFfactor
```

class RrcCls

Rrc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_roff_factor() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:TETRa:FILTer:RRC:ROFFfactor
value: float = driver.configure.vse.measurement.tetra.filterPy.rrc.get_roff_
↪factor()
```

Queries the roll-off factor of the filter used for pulse shaping for TETRA.

return
rolloff_factor: No help available

6.3.6.1.10 Xrt

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:XRT:ENABle
```

class XrtCls

Xrt commands group definition. 4 total commands, 1 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:ENABle
value: bool = driver.configure.vse.measurement.xrt.get_enable()
```

Enables receiving of VSE measurement results via the CMA-XRT100 configuration.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:ENABle
driver.configure.vse.measurement.xrt.set_enable(enable = False)
```

Enables receiving of VSE measurement results via the CMA-XRT100 configuration.

param enable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.vse.measurement.xrt.clone()
```

Subgroups

6.3.6.1.10.1 RfSettings

SCPI Command:

```
CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:CONNector
CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:FREQuency
CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:ENPower
```

class RfSettingsCls

RfSettings commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_connector() → XrtInputConnector

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:CONNector
value: enums.XrtInputConnector = driver.configure.vse.measurement.xrt.
↪rfSettings.get_connector()
```

Selects the input connector at the R&S CMW100 available for CMA-XRT100 configuration.

return
input_connector: RF1 | RF2 | RF3 | RF4 | RF5 | RF6 | RF7 | RF8

get_envelope_power() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:ENPower
value: float = driver.configure.vse.measurement.xrt.rfSettings.get_envelope_
↪power()
```

Sets the expected nominal power of the measured RF signal for CMA-XRT100 configuration. The allowed range depends on several other settings, for example on the selected connector and the external attenuation. For supported ranges, refer to the data sheet.

return
exp_nominal_power: Unit: dBm

get_frequency() → float

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:FREQuency
value: float = driver.configure.vse.measurement.xrt.rfSettings.get_frequency()
```

Sets the center frequency of the RF analyzer for CMA-XRT100 configuration.

return
frequency: Range: 7E+7 Hz to 6 GHz, Unit: Hz

set_connector(*input_connector: XrtInputConnector*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:CONNector
driver.configure.vse.measurement.xrt.rfSettings.set_connector(input_connector =␣
↪enums.XrtInputConnector.RF1)
```

Selects the input connector at the R&S CMW100 available for CMA-XRT100 configuration.

param input_connector
RF1 | RF2 | RF3 | RF4 | RF5 | RF6 | RF7 | RF8

set_envelope_power(*exp_nominal_power: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:ENPower
driver.configure.vse.measurement.xrt.rfSettings.set_envelope_power(exp_nominal_
↪power = 1.0)
```

Sets the expected nominal power of the measured RF signal for CMA-XRT100 configuration. The allowed range depends on several other settings, for example on the selected connector and the external attenuation. For supported ranges, refer to the data sheet.

param exp_nominal_power
Unit: dBm

set_frequency(*frequency: float*) → None

```
# SCPI: CONFigure:VSE:MEASurement<Instance>:XRT:RFSettings:FREQuency
driver.configure.vse.measurement.xrt.rfSettings.set_frequency(frequency = 1.0)
```

Sets the center frequency of the RF analyzer for CMA-XRT100 configuration.

param frequency
Range: 7E+7 Hz to 6 GHz, Unit: Hz

6.4 Display

SCPI Command:

```
DISPlay:FORMat
```

class DisplayCls

Display commands group definition. 12 total commands, 4 Subgroups, 1 group commands

get_format_py() → str

```
# SCPI: DISPlay:FORMat
value: str = driver.display.get_format_py()
```

No command help available

return
arg_0: No help available

set_format_py(*arg_0: str*) → None

```
# SCPI: DISPlay:FORMat
driver.display.set_format_py(arg_0 = r1)
```

No command help available

param arg_0
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.display.clone()
```

Subgroups

6.4.1 AfRf

class AfRfCls

AfRf commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.display.afRf.clone()
```

Subgroups

6.4.1.1 Measurement

class MeasurementCls

Measurement commands group definition. 4 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.display.afRf.measurement.clone()
```

Subgroups

6.4.1.1.1 Application

SCPI Command:

```
DISPlay:AFRF:MEASurement<Instance>:APPLication:SELect
```

class ApplicationCls

Application commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → SubTab

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:APPLication:SELect
value: enums.SubTab = driver.display.afRf.measurement.application.get_select()
```

Configures the display of the subtabs in the 'Analyzer' tab.

return
sub_tab: OVERview | TRIM | RFResults | AFResults | FMSTereo | FFT | MULTitone | OSC | TONes Shows the specified subtab.

set_select(*sub_tab: SubTab*) → None

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:APPLication:SELect
driver.display.afRf.measurement.application.set_select(sub_tab = enums.SubTab.
↪AFResults)
```

Configures the display of the subtabs in the 'Analyzer' tab.

param sub_tab
OVERview | TRIM | RFResults | AFResults | FMSTereo | FFT | MULTitone | OSC | TONes Shows the specified subtab.

6.4.1.1.2 Audio

class AudioCls

Audio 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.display.afRf.measurement.audio.clone()
```

Subgroups

6.4.1.1.2.1 Application

SCPI Command:

```
DISPlay:AFRF:MEASurement<Instance>:AUDio:APPLication:SELect
```

class ApplicationCls

Application commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → SubTabAudioMeas

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:AUDio:APPLication:SELect
value: enums.SubTabAudioMeas = driver.display.afRf.measurement.audio.
↪application.get_select()
```

Configures the display of the subtabs in the 'Audio' tab.

return
sub_tab: OVERview | TRIM | AFResults | FFT | OSC

set_select(*sub_tab: SubTabAudioMeas*) → None

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:AUDio:APPLication:SELect
driver.display.afRf.measurement.audio.application.set_select(sub_tab = enums.
↪SubTabAudioMeas.AFResults)
```

Configures the display of the subtabs in the 'Audio' tab.

param sub_tab
OVERview | TRIM | AFResults | FFT | OSC

6.4.1.1.3 Digital

class DigitalCls

Digital 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.display.afRf.measurement.digital.clone()
```

Subgroups

6.4.1.1.3.1 Application

SCPI Command:

```
DISPlay:AFRF:MEASurement<Instance>:DIGital:APPLication:SELect
```

class ApplicationCls

Application commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → SubTabDigitalMeas

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:DIGital:APPLication:SELect
value: enums.SubTabDigitalMeas = driver.display.afRf.measurement.digital.
↪application.get_select()
```

Selects the subtab to display in the 'Digital' tab.

return
sub_tab: OVERview | SINFo | BER OVERview 'Overview' subtab SINFo 'Signal Info' subtab BER 'BER' subtab

set_select(*sub_tab: SubTabDigitalMeas*) → None

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:DIGital:APPLication:SELect
driver.display.afRf.measurement.digital.application.set_select(sub_tab = enums.
↪SubTabDigitalMeas.BER)
```

Selects the subtab to display in the 'Digital' tab.

param sub_tab
OVERview | SINFo | BER OVERview 'Overview' subtab SINFo 'Signal Info' subtab BER 'BER' subtab

6.4.1.1.4 Routines

class RoutinesCls

Routines 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.display.afRf.measurement.routines.clone()
```

Subgroups

6.4.1.1.4.1 Application

SCPI Command:

```
DISPlay:AFRF:MEASurement<Instance>:ROUTines:APPLication:SELect
```

class ApplicationCls

Application commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → SubTabRoutines

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:ROUTines:APPLication:SELect
value: enums.SubTabRoutines = driver.display.afRf.measurement.routines.
↪application.get_select()
```

Configures the display of the subtabs in the 'Routines' tab.

return
sub_tab: CHARt | TABLe

set_select(*sub_tab: SubTabRoutines*) → None

```
# SCPI: DISPlay:AFRF:MEASurement<Instance>:ROUTines:APPLication:SELect
driver.display.afRf.measurement.routines.application.set_select(sub_tab = enums.
↪SubTabRoutines.CHARt)
```

Configures the display of the subtabs in the 'Routines' tab.

param sub_tab
CHARt | TABLe

6.4.2 GprfMeasurement

class GprfMeasurementCls

GprfMeasurement 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.display.gprfMeasurement.clone()
```

Subgroups

6.4.2.1 Acp

class AcpCls

Acp 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.display.gprfMeasurement.acp.clone()
```

Subgroups

6.4.2.1.1 Trace

SCPI Command:

```
DISPlay:GPRF:MEASurement<Instance>:ACP:TRACe
```

class TraceCls

Trace commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*trace_enable: bool*, *trace: TraceC*) → None

```
# SCPI: DISPlay:GPRF:MEASurement<Instance>:ACP:TRACe
driver.display.gprfMeasurement.acp.trace.set(trace_enable = False, trace =␣
↪enums.TraceC.AVERage)
```

Selects which set of results is displayed on the 'ACP' tab. Only one set of results is displayed at a time. Enabling one set disables the others.

param trace_enable
OFF | ON Disables or enables the set selected via Trace

param trace
CURRent | AVERage | MAXimum Selects the set of results to be enabled/disabled

6.4.2.2 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.display.gprfMeasurement.extPwrSensor.clone()
```

Subgroups

6.4.2.2.1 Application

SCPI Command:

```
DISPlay:GPRF:MEASurement<Instance>:EPSensor:APPLication:SELect
```

class ApplicationCls

Application commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → ExtPwrSensorApp

```
# SCPI: DISPlay:GPRF:MEASurement<Instance>:EPSensor:APPLication:SELect
value: enums.ExtPwrSensorApp = driver.display.gprfMeasurement.extPwrSensor.
↪application.get_select()
```

Configures the display of the 'Sensor' tab.

return
application: EPS | NRTZ Show 'EPS' subtab or 'NRT-Z' subtab

set_select(*application: ExtPwrSensorApp*) → None

```
# SCPI: DISPlay:GPRF:MEASurement<Instance>:EPSensor:APPLication:SELect
driver.display.gprfMeasurement.extPwrSensor.application.set_select(application␣
↪= enums.ExtPwrSensorApp.EPS)
```

Configures the display of the 'Sensor' tab.

param application
EPS | NRTZ Show 'EPS' subtab or 'NRT-Z' subtab

6.4.2.3 FftSpecAn

class FftSpecAnCls

FftSpecAn 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.display.gprfMeasurement.fftSpecAn.clone()
```

Subgroups

6.4.2.3.1 Trace

SCPI Command:

```
DISPlay:GPRF:MEASurement<Instance>:FFTSanalyzer:TRACe
```

class TraceCls

Trace commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*trace_enable: bool*, *trace: TraceB = None*) → None

```
# SCPI: DISPlay:GPRF:MEASurement<Instance>:FFTSanalyzer:TRACe
driver.display.gprfMeasurement.fftSpecAn.trace.set(trace_enable = False, trace␣
↪= enums.TraceB.AVERage)
```

Selects which traces and diagrams are displayed on the 'FFT Spectrum' tab.

param trace_enable
OFF | ON Disables or enables the trace/diagrams selected via Trace

param trace
CURRent | AVERage | MAXimum | MINimum | TDOMmain Selects the trace (current, average, max, min) or diagrams (time domain) to be enabled/disabled To enable or disable all traces and diagrams, omit the parameter.

6.4.2.4 Spectrum

class SpectrumCls

Spectrum 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.display.gprfMeasurement.spectrum.clone()
```

Subgroups

6.4.2.4.1 Application

class ApplicationCls

Application 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.display.gprfMeasurement.spectrum.application.clone()
```

Subgroups

6.4.2.4.1.1 Select

SCPI Command:

```
DISPlay:GPRF:MEASurement<Instance>:SPECtrum:APPLication:SELect
```

class SelectCls

Select commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SelectStruct

Response structure. Fields:

- Application: enums.SpecAnApp: FREQ | ZERO Show 'Frequency Sweep' subtab or 'Zero Span' subtab
- Fullscreen: bool: OFF | ON OFF: show result diagram with default size ON: maximize result diagram

get() → SelectStruct

```
# SCPI: DISPlay:GPRF:MEASurement<Instance>:SPECtrum:APPLication:SELect
value: SelectStruct = driver.display.gprfMeasurement.spectrum.application.
↪select.get()
```

Configures the display of the 'Spectrum Analyzer' tab.

return
structure: for return value, see the help for SelectStruct structure arguments.

set(*application: SpecAnApp*, *fullscreen: bool = None*) → None

```
# SCPI: DISPlay:GPRF:MEASurement<Instance>:SPECtrum:APPLication:SELect
driver.display.gprfMeasurement.spectrum.application.select.set(application =␣
↪enums.SpecAnApp.FREQ, fullscreen = False)
```

Configures the display of the 'Spectrum Analyzer' tab.

param application
FREQ | ZERO Show 'Frequency Sweep' subtab or 'Zero Span' subtab

param fullscreen
OFF | ON OFF: show result diagram with default size ON: maximize result diagram

6.4.2.4.2 Trace

SCPI Command:

```
DISPlay:GPRF:MEASurement<Instance>:SPECtrum:TRACe
```

class TraceCls

Trace commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*trace_enable: bool*, *trace: Statistic = None*) → None

```
# SCPI: DISPlay:GPRF:MEASurement<Instance>:SPECtrum:TRACe
driver.display.gprfMeasurement.spectrum.trace.set(trace_enable = False, trace =␣
↪enums.Statistic.AVERage)
```

Selects which traces are displayed on the 'Spectrum Analyzer' tab.

param trace_enable
OFF | ON Disables or enables the trace selected via Trace

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace to be enabled/disabled
To enable or disable all traces, omit the parameter.

6.4.3 Vse

class VseCls

Vse 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.display.vse.clone()
```

Subgroups

6.4.3.1 Measurement

class MeasurementCls

Measurement commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.display.vse.measurement.clone()
```

Subgroups

6.4.3.1.1 Application

SCPI Command:

```
DISPlay:VSE:MEASurement<Instance>:APPLication:SELect
```

class ApplicationCls

Application commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_select() → SubTabVseMeas

```
# SCPI: DISPlay:VSE:MEASurement<Instance>:APPLication:SELect
value: enums.SubTabVseMeas = driver.display.vse.measurement.application.get_
↪select()
```

Configures the display of the subtabs in the 'VSE' tab.

return
sub_tab: OVERview | RFResults | DEMod | SYMResults | PVTResults | NXDNresults | EYEDiagram | CONStellation | SYMDistr | LTE | SPECtrum

set_select(*sub_tab: SubTabVseMeas*) → None

```
# SCPI: DISPlay:VSE:MEASurement<Instance>:APPLication:SELect
driver.display.vse.measurement.application.set_select(sub_tab = enums.
↪SubTabVseMeas.CONStellation)
```

Configures the display of the subtabs in the 'VSE' tab.

param sub_tab
OVERview | RFResults | DEMod | SYMResults | PVTResults | NXDNresults | EYEDiagram | CONStellation | SYMDistr | LTE | SPECtrum

6.4.4 Window<Window>

RepCap Settings

```
# Range: Win1 .. Win32
rc = driver.display.window.repcap_window_get()
driver.display.window.repcap_window_set(repcap.Window.Win1)
```

class WindowCls

Window commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Window, default value after init: Window.Win1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.display.window.clone()
```

Subgroups

6.4.4.1 Select

SCPI Command:

```
DISPlay[:WINDow<1-n>]:SELect
```

class SelectCls

Select commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*window=Window.Default*) → None

```
# SCPI: DISPlay[:WINDow<1-n>]:SELect
driver.display.window.select.set(window = repcap.Window.Default)
```

No command help available

param window
optional repeated capability selector. Default value: Win1 (settable in the interface 'Window')

set_with_opc(*window=Window.Default*, *opc_timeout_ms: int = -1*) → None

6.5 FirmwareUpdate

SCPI Command:

```
FETCh:FWUPdate:VERSions
```

class FirmwareUpdateCls

FirmwareUpdate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_versions() → List[str]

```
# SCPI: FETCh:FWUPdate:VERSions
value: List[str] = driver.firmwareUpdate.get_versions()
```

No command help available

return
versions: No help available

6.6 FormatPy

class FormatPyCls

FormatPy commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.formatPy.clone()
```

Subgroups

6.6.1 Base

SCPI Command:

```
FORMat:BASE:BORDer
FORMat:BASE:DINTerchange
FORMat:BASE:SREGister
```

class BaseCls

Base commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_border() → ByteOrder

```
# SCPI: FORMat:BASE:BORDer
value: enums.ByteOrder = driver.formatPy.base.get_border()
```

No command help available

return
byte_order: No help available

get_dinterchange() → bool

```
# SCPI: FORMat:BASE:DINTerchange
value: bool = driver.formatPy.base.get_dinterchange()
```

No command help available

return
dif_format: No help available

get_sregister() → StatRegFormat

```
# SCPI: FORMat:BASE:SREGister
value: enums.StatRegFormat = driver.formatPy.base.get_sregister()
```

No command help available

return
status_register_format: No help available

set_border(*byte_order: ByteOrder*) → None

```
# SCPI: FORMat:BASE:BORDer
driver.formatPy.base.set_border(byte_order = enums.ByteOrder.NORMal)
```

No command help available

param byte_order
No help available

set_dinterchange(*dif_format: bool*) → None

```
# SCPI: FORMat:BASE:DINTerchange
driver.formatPy.base.set_dinterchange(dif_format = False)
```

No command help available

param dif_format
No help available

set_sregister(*status_register_format: StatRegFormat*) → None

```
# SCPI: FORMat:BASE:SREGister
driver.formatPy.base.set_sregister(status_register_format = enums.StatRegFormat.
↪ASCii)
```

No command help available

param status_register_format
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.formatPy.base.clone()
```

Subgroups

6.6.1.1 Data

SCPI Command:

```
FORMat:BASE[:DATA]
```

class DataCls

Data commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DataStruct

Response structure. Fields:

- Data_Type: enums.DataFormat: ASCii | REAL | BINary | HEXadecimal | OCTal ASCii Numeric data is transferred as ASCII bytes. Floating point numbers are transferred in scientific E notation. REAL Numeric data is transferred in a definite length block as IEEE floating point numbers, see 'Block data'. BINary | HEXadecimal | OCTal Numeric data is transferred in binary, hexadecimal or octal format.

- Data_Length: int: The meaning depends on the DataType as listed below. A zero returned by a query means that the default value is used. For ASCii Decimal places of floating point numbers. That means, number of 'b' digits in the scientific notation a.bbbbbbE+ccc. Default: 6 decimal places For REAL Length of floating point numbers in bits: 32 bits = 4 bytes, format #14... 64 bits = 8 bytes, format #18... Default: 64 bits For BINary, HEXadecimal, OCTal Minimum number of digits. If the number is longer, more digits are used. If it is shorter, leading zeros are added. Default: 0, no leading zeros

get() → DataStruct

```
# SCPI: FORMat:BASE[:DATA]
value: DataStruct = driver.formatPy.base.data.get()
```

Selects the format for numeric data transferred by the instrument, for example query results.

return
structure: for return value, see the help for DataStruct structure arguments.

set(*data_type: DataFormat*, *data_length: int = None*) → None

```
# SCPI: FORMat:BASE[:DATA]
driver.formatPy.base.data.set(data_type = enums.DataFormat.ASCii, data_length =␣
↪1)
```

Selects the format for numeric data transferred by the instrument, for example query results.

param data_type
ASCii | REAL | BINary | HEXadecimal | OCTal ASCii Numeric data is transferred as ASCII bytes. Floating point numbers are transferred in scientific E notation. REAL Numeric data is transferred in a definite length block as IEEE floating point numbers, see 'Block data'. BINary | HEXadecimal | OCTal Numeric data is transferred in binary, hexadecimal or octal format.

param data_length
The meaning depends on the DataType as listed below. A zero returned by a query means that the default value is used. For ASCii Decimal places of floating point numbers. That means, number of 'b' digits in the scientific notation a.bbbbbbE+ccc. Default: 6 decimal places For REAL Length of floating point numbers in bits: 32 bits = 4 bytes, format #14... 64 bits = 8 bytes, format #18... Default: 64 bits For BINary, HEXadecimal, OCTal Minimum number of digits. If the number is longer, more digits are used. If it is shorter, leading zeros are added. Default: 0, no leading zeros

6.7 GprfMeasurement

class GprfMeasurementCls

GprfMeasurement commands group definition. 198 total commands, 7 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.clone()
```

Subgroups

6.7.1 Acp

SCPI Command:

```
INITiate:GPRF:MEASurement<Instance>:ACP
STOP:GPRF:MEASurement<Instance>:ACP
ABORt:GPRF:MEASurement<Instance>:ACP
```

class AcpCls

Acp commands group definition. 38 total commands, 4 Subgroups, 3 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:ACP
driver.gprfMeasurement.acp.abort()
```

Stops the ACP measurement.

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>:ACP
driver.gprfMeasurement.acp.initiate()
```

Starts or continues the ACP measurement.

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>:ACP
driver.gprfMeasurement.acp.stop()
```

Pauses the ACP measurement.

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.gprfMeasurement.acp.clone()
```

Subgroups

6.7.1.1 Aclr

class AclrCls

Aclr commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.acp.aclr.clone()
```

Subgroups

6.7.1.1.1 Average

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:AVERage
READ:GPRF:MEASurement<Instance>:ACP:ACLR:AVERage
CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:AVERage
value: List[float or bool] = driver.gprfMeasurement.acp.aclr.average.calculate()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return

aclr: (float or boolean items) Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:AVERage
value: List[float] = driver.gprfMeasurement.acp.aclr.average.fetch()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:ACLR:AVERage
value: List[float] = driver.gprfMeasurement.acp.aclr.average.read()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

6.7.1.1.2 Current

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:CURRent
READ:GPRF:MEASurement<Instance>:ACP:ACLR:CURRent
CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:CURRent
value: List[float or bool] = driver.gprfMeasurement.acp.aclr.current.calculate()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: (float or boolean items) Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:CURRent
value: List[float] = driver.gprfMeasurement.acp.aclr.current.fetch()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:ACLR:CURRent
value: List[float] = driver.gprfMeasurement.acp.aclr.current.read()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

6.7.1.1.3 Maximum

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:MAXimum
READ:GPRF:MEASurement<Instance>:ACP:ACLR:MAXimum
CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:MAXimum
value: List[float or bool] = driver.gprfMeasurement.acp.aclr.maximum.calculate()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: (float or boolean items) Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:MAXimum
value: List[float] = driver.gprfMeasurement.acp.aclr.maximum.fetch()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:ACLR:MAXimum
value: List[float] = driver.gprfMeasurement.acp.aclr.maximum.read()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

6.7.1.1.4 StandardDev

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:SDEViation
READ:GPRF:MEASurement<Instance>:ACP:ACLR:SDEViation
CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:SDEViation
```

class StandardDevCls

StandardDev commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:ACLR:SDEViation
value: List[float or bool] = driver.gprfMeasurement.acp.aclr.standardDev.
↪calculate()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: (float or boolean items) Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:ACLR:SDEViation
value: List[float] = driver.gprfMeasurement.acp.aclr.standardDev.fetch()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:ACLR:SDEViation
value: List[float] = driver.gprfMeasurement.acp.aclr.standardDev.read()
```

Query the ACLR results. There are separate commands for the current, average, maximum and standard deviation values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
aclr: Comma-separated list of five ACLR values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dB

6.7.1.2 Obw

class ObwCls

Obw commands group definition. 9 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.acp.obw.clone()
```

Subgroups

6.7.1.2.1 Average

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:OBW:AVERage
READ:GPRF:MEASurement<Instance>:ACP:OBW:AVERage
CALCulate:GPRF:MEASurement<Instance>:ACP:OBW:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:OBW:AVERage
value: List[float or bool] = driver.gprfMeasurement.acp.obw.average.calculate()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: (float or boolean items) Unit: Hz

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:OBW:AVERage
value: List[float] = driver.gprfMeasurement.acp.obw.average.fetch()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: Unit: Hz

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:OBW:AVERage
value: List[float] = driver.gprfMeasurement.acp.obw.average.read()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: Unit: Hz

6.7.1.2.2 Current

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:OBW:CURRent
READ:GPRF:MEASurement<Instance>:ACP:OBW:CURRent
CALCulate:GPRF:MEASurement<Instance>:ACP:OBW:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:OBW:CURRent
value: List[float or bool] = driver.gprfMeasurement.acp.obw.current.calculate()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: (float or boolean items) Unit: Hz

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:OBW:CURRent
value: List[float] = driver.gprfMeasurement.acp.obw.current.fetch()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: Unit: Hz

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:OBW:CURRent
value: List[float] = driver.gprfMeasurement.acp.obw.current.read()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: Unit: Hz

6.7.1.2.3 Maximum

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:OBW:MAXimum
READ:GPRF:MEASurement<Instance>:ACP:OBW:MAXimum
CALCulate:GPRF:MEASurement<Instance>:ACP:OBW:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:OBW:MAXimum
value: List[float or bool] = driver.gprfMeasurement.acp.obw.maximum.calculate()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: (float or boolean items) Unit: Hz

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:OBW:MAXimum
value: List[float] = driver.gprfMeasurement.acp.obw.maximum.fetch()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: Unit: Hz

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:OBW:MAXimum
value: List[float] = driver.gprfMeasurement.acp.obw.maximum.read()
```

Query the OBW result. There are separate commands for the current, average and maximum value. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
obw: Unit: Hz

6.7.1.3 Power

class PowerCls

Power commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.acp.power.clone()
```

Subgroups

6.7.1.3.1 Average

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:POWer:AVERage
READ:GPRF:MEASurement<Instance>:ACP:POWer:AVERage
CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:AVERage
value: List[float or bool] = driver.gprfMeasurement.acp.power.average.
↪calculate()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: (float or boolean items) Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:POWer:AVERage
value: List[float] = driver.gprfMeasurement.acp.power.average.fetch()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:POWer:AVERage
value: List[float] = driver.gprfMeasurement.acp.power.average.read()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

6.7.1.3.2 Current

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:POWer:CURRent
READ:GPRF:MEASurement<Instance>:ACP:POWer:CURRent
CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:CURRent
value: List[float or bool] = driver.gprfMeasurement.acp.power.current.
↪calculate()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: (float or boolean items) Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:POWer:CURRent
value: List[float] = driver.gprfMeasurement.acp.power.current.fetch()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:POWer:CURRent
value: List[float] = driver.gprfMeasurement.acp.power.current.read()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

6.7.1.3.3 Maximum

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:POWer:MAXimum
READ:GPRF:MEASurement<Instance>:ACP:POWer:MAXimum
CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:MAXimum
value: List[float or bool] = driver.gprfMeasurement.acp.power.maximum.
↪calculate()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: (float or boolean items) Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:POWer:MAXimum
value: List[float] = driver.gprfMeasurement.acp.power.maximum.fetch()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:POWer:MAXimum
value: List[float] = driver.gprfMeasurement.acp.power.maximum.read()
```

Query the absolute power results. There are separate commands for the current, average and maximum values. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of five power values, one per channel: Channel -2, -1, 0, +1, +2 Unit: dBm

6.7.1.3.4 Minimum

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:POWer:MINimum
READ:GPRF:MEASurement<Instance>:ACP:POWer:MINimum
CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:MINimum
```

class MinimumCls

Minimum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:ACP:POWer:MINimum
value: List[float or bool] = driver.gprfMeasurement.acp.power.minimum.
↪calculate()
```

Query the minimum absolute power of the designated channel. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: (float or boolean items) Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:POWer:MINimum
value: List[float] = driver.gprfMeasurement.acp.power.minimum.fetch()
```

Query the minimum absolute power of the designated channel. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:ACP:POWer:MINimum
value: List[float] = driver.gprfMeasurement.acp.power.minimum.read()
```

Query the minimum absolute power of the designated channel. CALCulate commands return error indicators instead of measurement values.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Unit: dBm

6.7.1.4 State

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:STATe
value: enums.ResourceState = driver.gprfMeasurement.acp.state.fetch()
```

Queries the main ACP measurement state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.acp.state.clone()
```

Subgroups

6.7.1.4.1 All

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:ACP:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:ACP:STATe:ALL
value: List[enums.ResourceState] = driver.gprfMeasurement.acp.state.all.fetch()
```

Queries the main ACP measurement state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.7.2 ExtPwrSensor

SCPI Command:

```
INITiate:GPRF:MEASurement<Instance>:EPSensor
STOP:GPRF:MEASurement<Instance>:EPSensor
ABORt:GPRF:MEASurement<Instance>:EPSensor
FETCh:GPRF:MEASurement<Instance>:EPSensor:IDN
FETCh:GPRF:MEASurement<Instance>:EPSensor
READ:GPRF:MEASurement<Instance>:EPSensor
```

class ExtPwrSensorCls

ExtPwrSensor commands group definition. 8 total commands, 1 Subgroups, 6 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Current_Power: float: Sensor power in the last measurement interval Unit: dBm
- Average_Power: float: Average of all CurrentPower values within the last measurement cycle Unit: dBm
- Minimum_Power: float: Minimum CurrentPower value since the start of the measurement Unit: dBm
- Maximum_Power: float: Maximum CurrentPower value since the start of the measurement Unit: dBm
- Elapsed_Stat: int: Elapsed statistic count (progress bar) Range: 0 to configured statistic count

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:EPSensor
driver.gprfMeasurement.extPwrSensor.abort()
```

Stops the EPS measurement.

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.gprfMeasurement.extPwrSensor.fetch()
```

Return all EPS measurement results.

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.gprfMeasurement.extPwrSensor.get_idn()
```

Queries the identification string of the connected external power sensor.

return
idn: String parameter

initiate(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:EPSensor
driver.gprfMeasurement.extPwrSensor.initiate()
```

Starts or continues the EPS measurement.

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.gprfMeasurement.extPwrSensor.read()
```

Return all EPS measurement results.

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.gprfMeasurement.extPwrSensor.stop()
```

Pauses the EPS measurement.

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.gprfMeasurement.extPwrSensor.clone()
```

Subgroups

6.7.2.1 State

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe
value: enums.ResourceState = driver.gprfMeasurement.extPwrSensor.state.fetch()
```

Queries the main EPS measurement state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.extPwrSensor.state.clone()
```

Subgroups

6.7.2.1.1 All

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:EPSensor:STATe:ALL
value: List[enums.ResourceState] = driver.gprfMeasurement.extPwrSensor.state.
↪all.fetch()
```

Queries the main EPS measurement state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.7.3 FftSpecAn

SCPI Command:

```
INITiate:GPRF:MEASurement<Instance>:FFTSanalyzer
STOP:GPRF:MEASurement<Instance>:FFTSanalyzer
ABORt:GPRF:MEASurement<Instance>:FFTSanalyzer
```

class FftSpecAnCls

FftSpecAn commands group definition. 26 total commands, 7 Subgroups, 3 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:FFTSanalyzer
driver.gprfMeasurement.fftSpecAn.abort()
```

Stops the FFT spectrum analyzer.

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.gprfMeasurement.fftSpecAn.initiate()
```

Starts or continues the FFT spectrum analyzer.

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.gprfMeasurement.fftSpecAn.stop()
```

Pauses the FFT spectrum analyzer.

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.gprfMeasurement.fftSpecAn.clone()
```

Subgroups

6.7.3.1 Icomponent

SCPI Command:

```
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.gprfMeasurement.fftSpecAn.icomponent.fetch()
```

Query the contents of the time domain diagrams. There are separate commands for the I amplitudes and the Q amplitudes.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
idata: Comma-separated list of normalized I or Q amplitudes The number of values equals the configured FFT length. The order of the values corresponds to the I/Q vs. time diagram, from left to right.

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:I
value: List[float] = driver.gprfMeasurement.fftSpecAn.icomponent.read()
```

Query the contents of the time domain diagrams. There are separate commands for the I amplitudes and the Q amplitudes.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
idata: Comma-separated list of normalized I or Q amplitudes The number of values equals the configured FFT length. The order of the values corresponds to the I/Q vs. time diagram, from left to right.

6.7.3.2 Marker

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>
```

class MarkerCls

Marker commands group definition. 3 total commands, 2 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Absolute_Yvalue: float: Y-value of the marker Unit: dBm

fetch(*trace: Statistic*, *freq_value: float*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>
value: FetchStruct = driver.gprfMeasurement.fftSpecAn.marker.fetch(trace =␣
↪enums.Statistic.AVERage, freq_value = 1.0, marker = repcap.Marker.Nr1)
```

Moves marker number <no> to a specified x-value and returns the absolute coordinates. The x-value is understood as the difference of frequency to the center of frequency span. If <FreqValue> is set 0, the marker must be set to the center of frequency span. Absolute placement is used.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param freq_value
X-value for which the coordinates are queried Unit: Hz

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.fftSpecAn.marker.clone()
```

Subgroups

6.7.3.2.1 Absolute

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Yvalue: float: Y-value of the marker Unit: dBm

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>:ABSolute
value: FetchStruct = driver.gprfMeasurement.fftSpecAn.marker.absolute.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪marker = repcap.Marker.Nr1)
```

Queries the absolute coordinates of marker number <no>. Marker number one is the reference marker. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.2.2 Relative

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Dx_Value: float: Delta X value of the marker relative to the reference marker Unit: Hz
- Dy_Value: float: Delta Y value of the marker relative to the reference marker Unit: dB

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *markerOther=MarkerOther.Nr2*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:MARKer<nr>:RELative
value: FetchStruct = driver.gprfMeasurement.fftSpecAn.marker.relative.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪markerOther = repcap.MarkerOther.Nr2)
```

Queries the relative coordinates of marker number <no>. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace

param markerOther
optional repeated capability selector. Default value: Nr2

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.3 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.gprfMeasurement.fftSpecAn.peaks.clone()
```

Subgroups

6.7.3.3.1 Average

SCPI Command:

```
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: List[float]: Frequency of the peak Unit: Hz
- Level: List[float]: Level of the peak Unit: dBm

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:AVERage
value: ResultData = driver.gprfMeasurement.fftSpecAn.peaks.average.fetch()
```

Query the contents of the peak search result table. There are separate commands for the current and average spectrum traces. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}Peak 1, ..., {<Frequency>, <Level>}Peak 5

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.gprfMeasurement.fftSpecAn.peaks.average.read()
```

Query the contents of the peak search result table. There are separate commands for the current and average spectrum traces. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}Peak 1, ..., {<Frequency>, <Level>}Peak 5

return
structure: for return value, see the help for ResultData structure arguments.

6.7.3.3.2 Current

SCPI Command:

```
READ:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Frequency: List[float]: Frequency of the peak Unit: Hz
- Level: List[float]: Level of the peak Unit: dBm

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:PEAKs:CURRent
value: ResultData = driver.gprfMeasurement.fftSpecAn.peaks.current.fetch()
```

Query the contents of the peak search result table. There are separate commands for the current and average spectrum traces. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}Peak 1, ..., {<Frequency>, <Level>}Peak 5

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.gprfMeasurement.fftSpecAn.peaks.current.read()
```

Query the contents of the peak search result table. There are separate commands for the current and average spectrum traces. The results are returned in the following order: <Reliability>, {<Frequency>, <Level>}Peak 1, ..., {<Frequency>, <Level>}Peak 5

return
structure: for return value, see the help for ResultData structure arguments.

6.7.3.4 Power

class PowerCls

Power commands group definition. 9 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.fftSpecAn.power.clone()
```

Subgroups

6.7.3.4.1 Average

SCPI Command:

```
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.gprfMeasurement.fftSpecAn.power.average.fetch()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:AVERage
value: List[float] = driver.gprfMeasurement.fftSpecAn.power.average.read()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

6.7.3.4.2 Current

SCPI Command:

```
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.gprfMeasurement.fftSpecAn.power.current.fetch()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:CURRent
value: List[float] = driver.gprfMeasurement.fftSpecAn.power.current.read()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

6.7.3.4.3 Maximum

SCPI Command:

```
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.gprfMeasurement.fftSpecAn.power.maximum.fetch()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MAXimum
value: List[float] = driver.gprfMeasurement.fftSpecAn.power.maximum.read()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

6.7.3.4.4 Minimum

SCPI Command:

```
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.gprfMeasurement.fftSpecAn.power.minimum.fetch()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:MINimum
value: List[float] = driver.gprfMeasurement.fftSpecAn.power.minimum.read()
```

Query the contents of the spectrum diagram. There are separate commands for the current, average, minimum and maximum power traces.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 801 power values The power values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two results equals span/800. Unit: dBm

6.7.3.4.5 Xvalues

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:XVALues
```

class XvaluesCls

Xvalues commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:POWer:XVALues
value: List[float] = driver.gprfMeasurement.fftSpecAn.power.xvalues.fetch()
```

Queries the x-values of the spectrum diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
xvalues: Comma-separated list of 801 frequency values The frequency values cover the entire measured frequency span, from the lower end to the upper end. The frequency distance between two x-values equals span/800. Unit: Hz

6.7.3.5 Qcomponent

SCPI Command:

```
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.gprfMeasurement.fftSpecAn.qcomponent.fetch()
```

Query the contents of the time domain diagrams. There are separate commands for the I amplitudes and the Q amplitudes.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
qdata: Comma-separated list of normalized I or Q amplitudes The number of values equals the configured FFT length. The order of the values corresponds to the I/Q vs. time diagram, from left to right.

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:FFTSanalyzer:Q
value: List[float] = driver.gprfMeasurement.fftSpecAn.qcomponent.read()
```

Query the contents of the time domain diagrams. There are separate commands for the I amplitudes and the Q amplitudes.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
qdata: Comma-separated list of normalized I or Q amplitudes The number of values equals the configured FFT length. The order of the values corresponds to the I/Q vs. time diagram, from left to right.

6.7.3.6 State

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe
value: enums.ResourceState = driver.gprfMeasurement.fftSpecAn.state.fetch()
```

Queries the main FFT spectrum analyzer state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.fftSpecAn.state.clone()
```

Subgroups

6.7.3.6.1 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() → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:STATe:ALL
value: List[enums.ResourceState] = driver.gprfMeasurement.fftSpecAn.state.all.
↪fetch()
```

Queries the main FFT spectrum analyzer state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.7.3.7 Tdomain

class TdomainCls

Tdomain 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.gprfMeasurement.fftSpecAn.tdomain.clone()
```

Subgroups

6.7.3.7.1 Xvalues

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:TDOMain:XVALues
```

class XvaluesCls

Xvalues commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:FFTSanalyzer:TDOMain:XVALues
value: List[float] = driver.gprfMeasurement.fftSpecAn.tdomain.xvalues.fetch()
```

Queries the x-values of the time domain diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
xvalues: Comma-separated list of time values The number of values equals the configured FFT length. The order of the values corresponds to the I/Q vs. time diagram, from left to right. Unit: s

6.7.4 IqRecorder

SCPI Command:

```
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.gprfMeasurement.iqRecorder.abort()
```

Stops the I/Q recorder measurement.

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.gprfMeasurement.iqRecorder.fetch()
```

Return 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'.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
iq_samples: For ASCII format: I/Q amplitudes in alternating order (I_1,Q_1, ...,I_n,Q_n) , as voltages. For REAL format: Binary block data consisting of the parts listed in the table below. There are no commas within this parameter.

initiate(*save_to_iq_file: FileSave = None*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:IQRecorder
driver.gprfMeasurement.iqRecorder.initiate(save_to_iq_file = enums.FileSave.OFF)
```

Starts or continues the I/Q recorder measurement.

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 selection of the I/Q file, see method

RsCma.Configure.GprfMeasurement.IqRecorder.iqFile. OFF The results are only stored in the memory. ON The results are stored in the memory and in the file. ONLY The results are only stored in the file.

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQRecorder
value: List[float] = driver.gprfMeasurement.iqRecorder.read()
```

Return 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'.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
iq_samples: For ASCII format: I/Q amplitudes in alternating order (I_1,Q_1, ...,I_n,Q_n) , as voltages. For REAL format: Binary block data consisting of the parts listed in the table below. There are no commas within this parameter.

stop(*opc_timeout_ms: int = -1*) → None

```
# SCPI: STOP:GPRF:MEASurement<Instance>:IQRecorder
driver.gprfMeasurement.iqRecorder.stop()
```

Pauses the I/Q recorder measurement.

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.gprfMeasurement.iqRecorder.clone()
```

Subgroups

6.7.4.1 Bin

SCPI Command:

```
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.gprfMeasurement.iqRecorder.bin.fetch()
```

Retrieve the I/Q recorder results in binary format.

return
iq_samples: Binary block data, see 'ASCII and binary data formats'

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:IQRecorder:BIN
value: List[float] = driver.gprfMeasurement.iqRecorder.bin.read()
```

Retrieve the I/Q recorder results in binary format.

return
iq_samples: Binary block data, see 'ASCII and binary data formats'

6.7.4.2 Reliability

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:RELiability
```

class ReliabilityCls

Reliability commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → int

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:RELiability
value: int = driver.gprfMeasurement.iqRecorder.reliability.fetch()
```

Queries the reliability indicator for the I/Q recorder measurement, see 'Reliability indicator values'.

return
reliability_flag: Two equal values, separated by a comma (for example 0,0 for 'OK')

6.7.4.3 State

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe
value: enums.ResourceState = driver.gprfMeasurement.iqRecorder.state.fetch()
```

Queries the main I/Q recorder measurement state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.iqRecorder.state.clone()
```

Subgroups

6.7.4.3.1 All

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:STATe:ALL
value: List[enums.ResourceState] = driver.gprfMeasurement.iqRecorder.state.all.
↪fetch()
```

Queries the main I/Q recorder measurement state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.7.4.4 SymbolRate

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:IQRecorder:SRATe
```

class SymbolRateCls

SymbolRate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → float

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:IQRecorder:SRATe
value: float = driver.gprfMeasurement.iqRecorder.symbolRate.fetch()
```

Returns the maximum sample rate, resulting from the filter settings.

return
sample_rate: Unit: Hz

6.7.4.5 Talignment

SCPI Command:

```
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.gprfMeasurement.iqRecorder.talignment.fetch()
```

No command help available

Use RsCma.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.gprfMeasurement.iqRecorder.talignment.read()
```

No command help available

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
time_alignment: No help available

6.7.5 Nrt

SCPI Command:

```
INITiate:GPRF:MEASurement<Instance>:NRT
STOP:GPRF:MEASurement<Instance>:NRT
ABORt:GPRF:MEASurement<Instance>:NRT
FETCh:GPRF:MEASurement<Instance>:NRT:IDN
```

class NrtCls

Nrt commands group definition. 30 total commands, 3 Subgroups, 4 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:NRT
driver.gprfMeasurement.nrt.abort()
```

Stops the NRT-Z measurement.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

get_idn() → str

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:IDN
value: str = driver.gprfMeasurement.nrt.get_idn()
```

Queries the identification string of the connected external power sensor.

return
idn: String parameter

initiate(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INITiate:GPRF:MEASurement<Instance>:NRT
driver.gprfMeasurement.nrt.initiate()
```

Starts or continues the NRT-Z measurement.

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>:NRT
driver.gprfMeasurement.nrt.stop()
```

Pauses the NRT-Z measurement.

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.gprfMeasurement.nrt.clone()
```

Subgroups

6.7.5.1 Forward

class ForwardCls

Forward commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.nrt.forward.clone()
```

Subgroups

6.7.5.1.1 Average

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:AVERage
FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:AVERage
READ:GPRF:MEASurement<Instance>:NRT:FWARd:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Forward power Unit: dBm
- Pep: enums.ResultStatus: Unit: dBm
- Crest_Factor: enums.ResultStatus: Unit: dB
- Ccdf: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Forward power Unit: dBm
- Pep: float: Unit: dBm
- Crest_Factor: float: Unit: dB
- Ccdf: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:AVERage
value: CalculateStruct = driver.gprfMeasurement.nrt.forward.average.calculate()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:AVERage
value: ResultData = driver.gprfMeasurement.nrt.forward.average.fetch()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:FWARd:AVERage
value: ResultData = driver.gprfMeasurement.nrt.forward.average.read()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.1.2 Current

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:CURRent
FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:CURRent
READ:GPRF:MEASurement<Instance>:NRT:FWARd:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Forward power Unit: dBm
- Pep: enums.ResultStatus: Unit: dBm
- Crest_Factor: enums.ResultStatus: Unit: dB
- Ccdf: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Forward power Unit: dBm
- Pep: float: Unit: dBm
- Crest_Factor: float: Unit: dB
- Ccdf: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:CURRent
value: CalculateStruct = driver.gprfMeasurement.nrt.forward.current.calculate()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:CURRent
value: ResultData = driver.gprfMeasurement.nrt.forward.current.fetch()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:FWARd:CURRent
value: ResultData = driver.gprfMeasurement.nrt.forward.current.read()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.1.3 Maximum

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:MAXimum
FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:MAXimum
READ:GPRF:MEASurement<Instance>:NRT:FWARd:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Forward power Unit: dBm
- Pep: enums.ResultStatus: Unit: dBm
- Crest_Factor: enums.ResultStatus: Unit: dB
- Ccdf: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Forward power Unit: dBm
- Pep: float: Unit: dBm
- Crest_Factor: float: Unit: dB
- Ccdf: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:MAXimum
value: CalculateStruct = driver.gprfMeasurement.nrt.forward.maximum.calculate()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:MAXimum
value: ResultData = driver.gprfMeasurement.nrt.forward.maximum.fetch()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:FWARd:MAXimum
value: ResultData = driver.gprfMeasurement.nrt.forward.maximum.read()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.1.4 Minimum

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:MINimum
FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:MINimum
READ:GPRF:MEASurement<Instance>:NRT:FWARd:MINimum
```

class MinimumCls

Minimum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Forward power Unit: dBm
- Pep: enums.ResultStatus: Unit: dBm
- Crest_Factor: enums.ResultStatus: Unit: dB
- Ccdf: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Forward power Unit: dBm
- Pep: float: Unit: dBm
- Crest_Factor: float: Unit: dB
- Ccdf: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:FWARd:MINimum
value: CalculateStruct = driver.gprfMeasurement.nrt.forward.minimum.calculate()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:FWARd:MINimum
value: ResultData = driver.gprfMeasurement.nrt.forward.minimum.fetch()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:FWARd:MINimum
value: ResultData = driver.gprfMeasurement.nrt.forward.minimum.read()
```

Return the measurement results for the forward direction. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.2 Reverse

class ReverseCls

Reverse commands group definition. 12 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.nrt.reverse.clone()
```

Subgroups

6.7.5.2.1 Average

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:AVERage
FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:AVERage
READ:GPRF:MEASurement<Instance>:NRT:REVerse:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Reverse power or forward power Unit: dBm
- Return_Loss: enums.ResultStatus: Unit: dB
- Reflection: enums.ResultStatus: Unit: %
- Swr: enums.ResultStatus: Standing wave ratio

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Reverse power or forward power Unit: dBm
- Return_Loss: float: Unit: dB
- Reflection: float: Unit: %
- Swr: float: Standing wave ratio

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:AVERage
value: CalculateStruct = driver.gprfMeasurement.nrt.reverse.average.calculate()
```

Return the measurement results for the reverse direction.

INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:AVERage
value: ResultData = driver.gprfMeasurement.nrt.reverse.average.fetch()
```

Return the measurement results for the reverse direction.
INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:REVerse:AVERage
value: ResultData = driver.gprfMeasurement.nrt.reverse.average.read()
```

Return the measurement results for the reverse direction.
INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.2.2 Current

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:CURRent
FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:CURRent
READ:GPRF:MEASurement<Instance>:NRT:REVerse:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Reverse power or forward power Unit: dBm

- Return_Loss: enums.ResultStatus: Unit: dB
- Reflection: enums.ResultStatus: Unit: %
- Swr: enums.ResultStatus: Standing wave ratio

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Reverse power or forward power Unit: dBm
- Return_Loss: float: Unit: dB
- Reflection: float: Unit: %
- Swr: float: Standing wave ratio

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:CURRent
value: CalculateStruct = driver.gprfMeasurement.nrt.reverse.current.calculate()
```

Return the measurement results for the reverse direction.

INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return

structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:CURRent
value: ResultData = driver.gprfMeasurement.nrt.reverse.current.fetch()
```

Return the measurement results for the reverse direction.

INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return

structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:REVerse:CURRent
value: ResultData = driver.gprfMeasurement.nrt.reverse.current.read()
```

Return the measurement results for the reverse direction.
INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.2.3 Maximum

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:MAXimum
FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:MAXimum
READ:GPRF:MEASurement<Instance>:NRT:REVerse:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Reverse power or forward power Unit: dBm
- Return_Loss: enums.ResultStatus: Unit: dB
- Reflection: enums.ResultStatus: Unit: %
- Swr: enums.ResultStatus: Standing wave ratio

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Reverse power or forward power Unit: dBm
- Return_Loss: float: Unit: dB
- Reflection: float: Unit: %
- Swr: float: Standing wave ratio

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:MAXimum
value: CalculateStruct = driver.gprfMeasurement.nrt.reverse.maximum.calculate()
```

Return the measurement results for the reverse direction.
INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.

- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:MAXimum
value: ResultData = driver.gprfMeasurement.nrt.reverse.maximum.fetch()
```

Return the measurement results for the reverse direction.
INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:REVerse:MAXimum
value: ResultData = driver.gprfMeasurement.nrt.reverse.maximum.read()
```

Return the measurement results for the reverse direction.
INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.2.4 Minimum

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:MINimum
FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:MINimum
READ:GPRF:MEASurement<Instance>:NRT:REVerse:MINimum
```

class MinimumCls

Minimum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: enums.ResultStatus: Reverse power or forward power Unit: dBm
- Return_Loss: enums.ResultStatus: Unit: dB
- Reflection: enums.ResultStatus: Unit: %
- Swr: enums.ResultStatus: Standing wave ratio

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Power: float: Reverse power or forward power Unit: dBm
- Return_Loss: float: Unit: dB
- Reflection: float: Unit: %
- Swr: float: Standing wave ratio

calculate() → CalculateStruct

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:NRT:REVerse:MINimum
value: CalculateStruct = driver.gprfMeasurement.nrt.reverse.minimum.calculate()
```

Return the measurement results for the reverse direction.

INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return

structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:REVerse:MINimum
value: ResultData = driver.gprfMeasurement.nrt.reverse.minimum.fetch()
```

Return the measurement results for the reverse direction.

INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return

structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GPRF:MEASurement<Instance>:NRT:REVerse:MINimum
value: ResultData = driver.gprfMeasurement.nrt.reverse.minimum.read()
```

Return the measurement results for the reverse direction.
INTRO_CMD_HELP: The meaning of the result <Power> depends on the value set via the command method RsCma.Configure.GprfMeasurement.Nrt.Forward.Value.enable:

- FPWR or PEP: <Power> is the reverse power.
- CFAC or CCDF: <Power> is the forward power.

CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.7.5.3 State

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:NRT:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:STATe
value: enums.ResourceState = driver.gprfMeasurement.nrt.state.fetch()
```

Queries the main NRT-Z measurement state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.nrt.state.clone()
```

Subgroups

6.7.5.3.1 All

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:NRT:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:NRT:STATe:ALL
value: List[enums.ResourceState] = driver.gprfMeasurement.nrt.state.all.fetch()
```

Queries the main NRT-Z measurement state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.7.6 Power

SCPI Command:

```
INITiate:GPRF:MEASurement<Instance>:POWer
STOP:GPRF:MEASurement<Instance>:POWer
ABORt:GPRF:MEASurement<Instance>:POWer
```

class PowerCls

Power commands group definition. 27 total commands, 8 Subgroups, 3 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:POWer
driver.gprfMeasurement.power.abort()
```

Stops the power measurement.

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.gprfMeasurement.power.initiate()
```

Starts or continues the power measurement.

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.gprfMeasurement.power.stop()
```

Pauses the power measurement.

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.gprfMeasurement.power.clone()
```

Subgroups

6.7.6.1 Average

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:POWer:AVERage
FETCh:GPRF:MEASurement<Instance>:POWer:AVERage
READ:GPRF:MEASurement<Instance>:POWer:AVERage
```

class AverageCls

Average commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[float or bool] = driver.gprfMeasurement.power.average.calculate()
```

Query the 'Power Average RMS' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_average_rms: (float or boolean items) 'Power Average RMS' result Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[float] = driver.gprfMeasurement.power.average.fetch()
```

Query the 'Power Average RMS' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_average_rms: 'Power Average RMS' result Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:AVERage
value: List[float] = driver.gprfMeasurement.power.average.read()
```

Query the 'Power Average RMS' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_average_rms: 'Power Average RMS' result Unit: dBm

6.7.6.2 Current

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:POWer:CURRent
FETCh:GPRF:MEASurement<Instance>:POWer:CURRent
READ:GPRF:MEASurement<Instance>:POWer:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[float or bool] = driver.gprfMeasurement.power.current.calculate()
```

Query the 'Power Current RMS' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_rms: (float or boolean items) 'Power Current RMS' result Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[float] = driver.gprfMeasurement.power.current.fetch()
```

Query the 'Power Current RMS' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_rms: 'Power Current RMS' result Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:CURRent
value: List[float] = driver.gprfMeasurement.power.current.read()
```

Query the 'Power Current RMS' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_rms: 'Power Current RMS' result Unit: dBm

6.7.6.3 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.gprfMeasurement.power.elapsedStats.fetch()
```

Returns the reliability indicator and the number of elapsed measurement intervals.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return

stat_count: Number of elapsed measurement intervals. Range: 0 to configured statistic count

6.7.6.4 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.gprfMeasurement.power.maximum.clone()
```

Subgroups

6.7.6.4.1 Current

SCPI Command:

```
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[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[float or bool] = driver.gprfMeasurement.power.maximum.current.
↪calculate()
```

Query the 'Power Current Max.' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_max: (float or boolean items) 'Power Current Max.' result Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[float] = driver.gprfMeasurement.power.maximum.current.fetch()
```

Query the 'Power Current Max.' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_max: 'Power Current Max.' result Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MAXimum:CURRent
value: List[float] = driver.gprfMeasurement.power.maximum.current.read()
```

Query the 'Power Current Max.' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_max: 'Power Current Max.' result Unit: dBm

6.7.6.5 Minimum

class MinimumCls

Minimum commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.power.minimum.clone()
```

Subgroups

6.7.6.5.1 Current

SCPI Command:

```
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[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[float or bool] = driver.gprfMeasurement.power.minimum.current.
↪calculate()
```

Query the 'Power Current Min.' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_min: (float or boolean items) 'Power Current Min.' result Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[float] = driver.gprfMeasurement.power.minimum.current.fetch()
```

Query the 'Power Current Min.' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_min: 'Power Current Min.' result Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:MINimum:CURRent
value: List[float] = driver.gprfMeasurement.power.minimum.current.read()
```

Query the 'Power Current Min.' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_current_min: 'Power Current Min.' result Unit: dBm

6.7.6.6 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.gprfMeasurement.power.peak.clone()
```

Subgroups

6.7.6.6.1 Maximum

SCPI Command:

```
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[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[float or bool] = driver.gprfMeasurement.power.peak.maximum.
↪calculate()
```

Query the 'Power Maximum' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_maximum_max: (float or boolean items) 'Power Maximum' result Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[float] = driver.gprfMeasurement.power.peak.maximum.fetch()
```

Query the 'Power Maximum' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_maximum_max: 'Power Maximum' result Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:PEAK:MAXimum
value: List[float] = driver.gprfMeasurement.power.peak.maximum.read()
```

Query the 'Power Maximum' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_maximum_max: 'Power Maximum' result Unit: dBm

6.7.6.6.2 Minimum

SCPI Command:

```
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[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[float or bool] = driver.gprfMeasurement.power.peak.minimum.
↪calculate()
```

Query the 'Power Minimum' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_minimum_min: (float or boolean items) 'Power Minimum' result Unit: dBm

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[float] = driver.gprfMeasurement.power.peak.minimum.fetch()
```

Query the 'Power Minimum' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_minimum_min: 'Power Minimum' result Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:PEAK:MINimum
value: List[float] = driver.gprfMeasurement.power.peak.minimum.read()
```

Query the 'Power Minimum' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_minimum_min: 'Power Minimum' result Unit: dBm

6.7.6.7 StandardDev

SCPI Command:

```
CALCulate:GPRF:MEASurement<Instance>:POWer:SDEViation
FETCh:GPRF:MEASurement<Instance>:POWer:SDEViation
READ:GPRF:MEASurement<Instance>:POWer:SDEViation
```

class StandardDevCls

StandardDev commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → List[float]

```
# SCPI: CALCulate:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[float or bool] = driver.gprfMeasurement.power.standardDev.
↪calculate()
```

Query the 'Standard Deviation' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_std_dev_cur: (float or boolean items) 'Standard Deviation' result Unit: dB

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[float] = driver.gprfMeasurement.power.standardDev.fetch()
```

Query the 'Standard Deviation' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_std_dev_cur: 'Standard Deviation' result Unit: dB

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:POWer:SDEViation
value: List[float] = driver.gprfMeasurement.power.standardDev.read()
```

Query the 'Standard Deviation' result. CALCulate commands return an error indicator instead of a power value.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_std_dev_cur: 'Standard Deviation' result Unit: dB

6.7.6.8 State

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:POWer:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:STATe
value: enums.ResourceState = driver.gprfMeasurement.power.state.fetch()
```

Queries the main power measurement state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.power.state.clone()
```

Subgroups

6.7.6.8.1 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() → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:POWer:STATe:ALL
value: List[enums.ResourceState] = driver.gprfMeasurement.power.state.all.
↪fetch()
```

Queries the main power measurement state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.7.7 Spectrum

SCPI Command:

```
INITiate:GPRF:MEASurement<Instance>:SPECtrum
STOP:GPRF:MEASurement<Instance>:SPECtrum
ABORt:GPRF:MEASurement<Instance>:SPECtrum
```

class SpectrumCls

Spectrum commands group definition. 56 total commands, 11 Subgroups, 3 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:GPRF:MEASurement<Instance>:SPECtrum
driver.gprfMeasurement.spectrum.abort()
```

Stops the spectrum analyzer.

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.gprfMeasurement.spectrum.initiate()
```

Starts or continues the spectrum analyzer.

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.gprfMeasurement.spectrum.stop()
```

Pauses the spectrum analyzer.

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.gprfMeasurement.spectrum.clone()
```

Subgroups

6.7.7.1 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.gprfMeasurement.spectrum.average.clone()
```

Subgroups

6.7.7.1.1 Average

SCPI Command:

```
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.gprfMeasurement.spectrum.average.average.fetch()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:AVERage
value: List[float] = driver.gprfMeasurement.spectrum.average.average.read()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.1.2 Current

SCPI Command:

```
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.gprfMeasurement.spectrum.average.current.fetch()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:CURRent
value: List[float] = driver.gprfMeasurement.spectrum.average.current.read()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.1.3 Maximum

SCPI Command:

```
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.gprfMeasurement.spectrum.average.maximum.fetch()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MAXimum
value: List[float] = driver.gprfMeasurement.spectrum.average.maximum.read()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.1.4 Minimum

SCPI Command:

```
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.gprfMeasurement.spectrum.average.minimum.fetch()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:AVERage:MINimum
value: List[float] = driver.gprfMeasurement.spectrum.average.minimum.read()
```

Query the result traces calculated with the 'Average' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.2 FreqSweep

class FreqSweepCls

FreqSweep 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.gprfMeasurement.spectrum.freqSweep.clone()
```

Subgroups

6.7.7.2.1 Xvalues

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:XVALues
```

class XvaluesCls

Xvalues commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:FSWeep:XVALues
value: List[float] = driver.gprfMeasurement.spectrum.freqSweep.xvalues.fetch()
```

Queries the x-values of the result traces in 'Frequency Sweep' mode.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
xvalues: Comma-separated list of 1001 frequency values Range: 50 Hz to 6 GHz, Unit: Hz

6.7.7.3 Frequency

class FrequencyCls

Frequency 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.gprfMeasurement.spectrum.frequency.clone()
```

Subgroups

6.7.7.3.1 Marker

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>
```

class MarkerCls

Marker commands group definition. 3 total commands, 2 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Absolute_Yvalue: float: Y-value of the marker Unit: dBm

fetch(*trace: Statistic*, *freq_value: float*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>
value: FetchStruct = driver.gprfMeasurement.spectrum.frequency.marker.
↪fetch(trace = enums.Statistic.AVERage, freq_value = 1.0, marker = repcap.
↪Marker.Nr1)
```

Moves marker number <no> to a specified x-value and returns the absolute coordinates in frequency sweep mode. Absolute placement is used. Marker number one is the reference marker. Select the trace to be evaluated and the x-value.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param freq_value
X-value for which the coordinates are queried Select a value within the measured span. Unit: Hz

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.spectrum.frequency.marker.clone()
```

Subgroups

6.7.7.3.1.1 Absolute

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: Hz
- Yvalue: float: Y-value of the marker Unit: dBm

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:ABSolute
value: FetchStruct = driver.gprfMeasurement.spectrum.frequency.marker.absolute.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪marker = repcap.Marker.Nr1)
```

Queries the absolute coordinates of marker number <no> in frequency sweep mode. Marker number one is the reference marker. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position. To configure a range for the action MAXV, see method RsCma.Configure.GprfMeasurement.Spectrum.Frequency.Marker.Range.set.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN | MAXV Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace MAXV Search the absolute maximum within a defined range of the trace

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.7.3.1.2 Relative

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Dx_Value: float: Delta X value of the marker relative to the reference marker Unit: Hz
- Dy_Value: float: Delta Y value of the marker relative to the reference marker Unit: dB

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *markerOther=MarkerOther.Nr2*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:FREQuency:MARKer<nr>:RELative
value: FetchStruct = driver.gprfMeasurement.spectrum.frequency.marker.relative.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪markerOther = repcap.MarkerOther.Nr2)
```

Queries the relative coordinates of marker number <no> in frequency sweep mode. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position. To configure a range for the action MAXV, see method RsCma.Configure.GprfMeasurement.Spectrum.Frequency.Marker.Range.set.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN | MAXV Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace MAXV Search the absolute maximum within a defined range of the trace

param markerOther
optional repeated capability selector. Default value: Nr2

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.7.4 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.gprfMeasurement.spectrum.maximum.clone()
```

Subgroups

6.7.7.4.1 Average

SCPI Command:

```
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.gprfMeasurement.spectrum.maximum.average.fetch()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:AVERage
value: List[float] = driver.gprfMeasurement.spectrum.maximum.average.read()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.4.2 Current

SCPI Command:

```
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.gprfMeasurement.spectrum.maximum.current.fetch()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:CURRent
value: List[float] = driver.gprfMeasurement.spectrum.maximum.current.read()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.4.3 Maximum

SCPI Command:

```
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.gprfMeasurement.spectrum.maximum.maximum.fetch()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MAXimum
value: List[float] = driver.gprfMeasurement.spectrum.maximum.maximum.read()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.4.4 Minimum

SCPI Command:

```
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.gprfMeasurement.spectrum.maximum.minimum.fetch()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MAXimum:MINimum
value: List[float] = driver.gprfMeasurement.spectrum.maximum.minimum.read()
```

Query the result traces calculated with the 'MaxPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.5 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.gprfMeasurement.spectrum.minimum.clone()
```

Subgroups

6.7.7.5.1 Average

SCPI Command:

```
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.gprfMeasurement.spectrum.minimum.average.fetch()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:AVERage
value: List[float] = driver.gprfMeasurement.spectrum.minimum.average.read()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.5.2 Current

SCPI Command:

```
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.gprfMeasurement.spectrum.minimum.current.fetch()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:CURRent
value: List[float] = driver.gprfMeasurement.spectrum.minimum.current.read()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.5.3 Maximum

SCPI Command:

```
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.gprfMeasurement.spectrum.minimum.maximum.fetch()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MAXimum
value: List[float] = driver.gprfMeasurement.spectrum.minimum.maximum.read()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.5.4 Minimum

SCPI Command:

```
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.gprfMeasurement.spectrum.minimum.minimum.fetch()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:MINimum:MINimum
value: List[float] = driver.gprfMeasurement.spectrum.minimum.minimum.read()
```

Query the result traces calculated with the 'MinPeak' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.6 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.gprfMeasurement.spectrum.referenceMarker.clone()
```

Subgroups

6.7.7.6.1 Npeak

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:NPEak
```

class NpeakCls

Npeak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Xvalue: float: No parameter help available
- Yvalue: float: No parameter help available

fetch(*detector: Detector*, *statistic: Statistic*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:NPEak
value: FetchStruct = driver.gprfMeasurement.spectrum.referenceMarker.npeak.
↪fetch(detector = enums.Detector.AUTopeak, statistic = enums.Statistic.AVERage)
```

No command help available

param detector
No help available

param statistic
No help available

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.7.6.2 Speak

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:SPEak
```

class SpeakCls

Speak commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Xvalue: float: No parameter help available
- Yvalue: float: No parameter help available

fetch(*detector: Detector*, *statistic: Statistic*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:REFMarker:SPEak
value: FetchStruct = driver.gprfMeasurement.spectrum.referenceMarker.speak.
↪fetch(detector = enums.Detector.AUTopeak, statistic = enums.Statistic.AVERage)
```

No command help available

param detector
No help available

param statistic
No help available

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.7.7 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.gprfMeasurement.spectrum.rms.clone()
```

Subgroups

6.7.7.7.1 Average

SCPI Command:

```
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.gprfMeasurement.spectrum.rms.average.fetch()
```

Query the result traces calculated with the ‘RMS’ detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:AVERage
value: List[float] = driver.gprfMeasurement.spectrum.rms.average.read()
```

Query the result traces calculated with the 'RMS' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.7.2 Current

SCPI Command:

```
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.gprfMeasurement.spectrum.rms.current.fetch()
```

Query the result traces calculated with the 'RMS' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:CURRent
value: List[float] = driver.gprfMeasurement.spectrum.rms.current.read()
```

Query the result traces calculated with the 'RMS' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.7.3 Maximum

SCPI Command:

```
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.gprfMeasurement.spectrum.rms.maximum.fetch()
```

Query the result traces calculated with the 'RMS' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MAXimum
value: List[float] = driver.gprfMeasurement.spectrum.rms.maximum.read()
```

Query the result traces calculated with the 'RMS' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.7.4 Minimum

SCPI Command:

```
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.gprfMeasurement.spectrum.rms.minimum.fetch()
```

Query the result traces calculated with the 'RMS' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:RMS:MINimum
value: List[float] = driver.gprfMeasurement.spectrum.rms.minimum.read()
```

Query the result traces calculated with the 'RMS' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.8 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.gprfMeasurement.spectrum.sample.clone()
```

Subgroups

6.7.7.8.1 Average

SCPI Command:

```
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.gprfMeasurement.spectrum.sample.average.fetch()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:AVERage
value: List[float] = driver.gprfMeasurement.spectrum.sample.average.read()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.8.2 Current

SCPI Command:

```
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.gprfMeasurement.spectrum.sample.current.fetch()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:CURRent
value: List[float] = driver.gprfMeasurement.spectrum.sample.current.read()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.8.3 Maximum

SCPI Command:

```
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.gprfMeasurement.spectrum.sample.maximum.fetch()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MAXimum
value: List[float] = driver.gprfMeasurement.spectrum.sample.maximum.read()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.8.4 Minimum

SCPI Command:

```
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.gprfMeasurement.spectrum.sample.minimum.fetch()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

read() → List[float]

```
# SCPI: READ:GPRF:MEASurement<Instance>:SPECtrum:SAMPle:MINimum
value: List[float] = driver.gprfMeasurement.spectrum.sample.minimum.read()
```

Query the result traces calculated with the 'Sample' detector. The current, average, minimum and maximum traces can be retrieved.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Comma-separated list of 1001 power values Unit: dBm

6.7.7.9 State

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe
value: enums.ResourceState = driver.gprfMeasurement.spectrum.state.fetch()
```

Queries the main spectrum analyzer state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.spectrum.state.clone()
```

Subgroups

6.7.7.9.1 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() → List[ResourceState]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:STATe:ALL
value: List[enums.ResourceState] = driver.gprfMeasurement.spectrum.state.all.
↪fetch()
```

Queries the main spectrum analyzer state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: No help available

6.7.7.10 Tgenerator

class TgeneratorCls

Tgenerator 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.gprfMeasurement.spectrum.tgenerator.clone()
```

Subgroups

6.7.7.10.1 RefDataAvailable

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:RDAVailable
```

class RefDataAvailableCls

RefDataAvailable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → bool

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:TGENerator:RDAVailable
value: bool = driver.gprfMeasurement.spectrum.tgenerator.refDataAvailable.
↪fetch()
```

Queries whether valid calibration results are available (ON) or not (OFF) .

return
ref_data_state: OFF | ON

6.7.7.11 ZeroSpan

class ZeroSpanCls

ZeroSpan 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.gprfMeasurement.spectrum.zeroSpan.clone()
```

Subgroups

6.7.7.11.1 Marker

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>
```

class MarkerCls

Marker commands group definition. 3 total commands, 2 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: s
- Absolute_Yvalue: float: Y-value of the marker Unit: dBm

fetch(*trace: Statistic*, *freq_value: float*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>
value: FetchStruct = driver.gprfMeasurement.spectrum.zeroSpan.marker.
↪fetch(trace = enums.Statistic.AVERage, freq_value = 1.0, marker = repcap.
↪Marker.Nr1)
```

Moves marker number <no> to a specified x-value and returns the absolute coordinates in zero span mode. Absolute placement is used. Marker number one is the reference marker. Select the trace to be evaluated and the x-value.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param freq_value
X-value for which the coordinates are queried Range: 0 s to sweep time, Unit: s

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.gprfMeasurement.spectrum.zeroSpan.marker.clone()
```

Subgroups

6.7.7.11.1.1 Absolute

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Xvalue: float: X-value of the marker Unit: s
- Yvalue: float: Y-value of the marker Unit: dBm

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *marker=Marker.Nr1*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:ABSolute
value: FetchStruct = driver.gprfMeasurement.spectrum.zeroSpan.marker.absolute.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪marker = repcap.Marker.Nr1)
```

Queries the absolute coordinates of marker number <no> in zero span mode. Marker number one is the reference marker. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position. To configure a range for the action MAXV, see method RsCma.Configure.GprfMeasurement.Spectrum.ZeroSpan.Marker.Range.set.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN | MAXV Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace MAXV Search the absolute maximum within a defined range of the trace

param marker
optional repeated capability selector. Default value: Nr1

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.7.11.1.2 Relative

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Dx_Value: float: Delta X value of the marker relative to the reference marker Unit: s
- Dy_Value: float: Delta Y value of the marker relative to the reference marker Unit: dB

fetch(*trace: Statistic*, *function: MarkerFunction = None*, *markerOther=MarkerOther.Nr2*) → FetchStruct

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:MARKer<nr>:RELative
value: FetchStruct = driver.gprfMeasurement.spectrum.zeroSpan.marker.relative.
↪fetch(trace = enums.Statistic.AVERage, function = enums.MarkerFunction.MAX,␣
↪markerOther = repcap.MarkerOther.Nr2)
```

Queries the relative coordinates of marker number <no> in zero span mode. Select the trace to be evaluated. Optionally, you can perform a marker action before reading the position. To configure a range for the action MAXV, see method RsCma. Configure.GprfMeasurement.Spectrum.ZeroSpan.Marker.Range.set.

param trace
CURRent | AVERage | MAXimum | MINimum Selects the trace type

param function
MIN | MAX | MAXL | MAXR | MAXN | MAXV Marker action to be performed before the query MIN Search the absolute minimum of the entire trace MAX Search the absolute maximum of the entire trace MAXL Search the absolute maximum to the left of the current marker position MAXR Search the absolute maximum to the right of the current marker position MAXN Search the next lower peak of the entire trace MAXV Search the absolute maximum within a defined range of the trace

param markerOther
optional repeated capability selector. Default value: Nr2

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.7.11.2 Xvalues

SCPI Command:

```
FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:XVALues
```

class XvaluesCls

Xvalues commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[float]

```
# SCPI: FETCh:GPRF:MEASurement<Instance>:SPECtrum:ZSPan:XVALues
value: List[float] = driver.gprfMeasurement.spectrum.zeroSpan.xvalues.fetch()
```

Queries the x-values of the result traces in 'Zero Span' mode.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
xvalues: Comma-separated list of 1001 time values Unit: s

6.8 HardCopy

SCPI Command:

```
HCOPy:DATA
HCOPy:FILE
```

class HardCopyCls

HardCopy commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_data() → bytes

```
# SCPI: HCOPy:DATA
value: bytes = driver.hardCopy.get_data()
```

Captures a screenshot and returns the result in block data format, see also 'Block data'. It is recommended to 'switch on' the display before sending this command, see method RsCma.System.Display.update.

return
data: Screenshot in 488.2 block data format

set_file(*filename: str*) → None

```
# SCPI: HCOPy:FILE
driver.hardCopy.set_file(filename = '1')
```

Captures a screenshot and stores it to the specified file. It is recommended to 'switch on' the display before sending this command, see method RsCma.System.Display.update.

param filename
String parameter specifying the absolute path and name of the file. The file extension is added automatically according to the configured format (see method RsCma.HardCopy.Device.formatPy) .

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.hardCopy.clone()
```

Subgroups

6.8.1 Device

SCPI Command:

```
HCOPy:DEVice:FORMat
```

class DeviceCls

Device commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_format_py() → ScreenshotFormat

```
# SCPI: HCOPy:DEVice:FORMat
value: enums.ScreenshotFormat = driver.hardCopy.device.get_format_py()
```

Specifies the format of screenshots created via a HCOPy command.

return
file_formats: BMP | JPG | PNG

set_format_py(*file_formats: ScreenshotFormat*) → None

```
# SCPI: HCOPy:DEVice:FORMat
driver.hardCopy.device.set_format_py(file_formats = enums.ScreenshotFormat.BMP)
```

Specifies the format of screenshots created via a HCOPy command.

param file_formats
BMP | JPG | PNG

6.9 Init

class InitCls

Init 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.init.clone()
```

Subgroups

6.9.1 Vse

class VseCls

Vse 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.init.vse.clone()
```

Subgroups

6.9.1.1 Measurement

SCPI Command:

```
INIT:VSE:MEASurement<Instance>
```

class MeasurementCls

Measurement commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INIT:VSE:MEASurement<Instance>
driver.init.vse.measurement.set()
```

Starts or continues the measurement.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.10 Instrument

SCPI Command:

```
INSTrument:NSELect
```

class InstrumentCls

Instrument commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_nselect() → int

```
# SCPI: INSTrument:NSELect
value: int = driver.instrument.get_nselect()
```

No command help available

return
arg_0: No help available

set_nselect(*arg_0: int*) → None

```
# SCPI: INSTrument:NSELect
driver.instrument.set_nselect(arg_0 = 1)
```

No command help available

param arg_0
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.instrument.clone()
```

Subgroups

6.10.1 Select

SCPI Command:

```
INSTrument[:SELect]
```

class SelectCls

Select commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → str

```
# SCPI: INSTrument[:SELect]
value: str = driver.instrument.select.get_value()
```

No command help available

return
instrument: No help available

set_value(*instrument: str*) → None

```
# SCPI: INSTrument[:SELect]
driver.instrument.select.set_value(instrument = r1)
```

No command help available

param instrument
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.instrument.select.clone()
```

Subgroups

6.10.1.1 Dstrategy

SCPI Command:

```
INSTrument[:SELect]:DSTRategy
```

class DstrategyCls

Dstrategy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DstrategyStruct

Response structure. Fields:

- Arg_0: enums.OperationMode: No parameter help available
- Arg_1: enums.Dstrategy: No parameter help available

get() → DstrategyStruct

```
# SCPI: INSTrument[:SELect]:DSTRategy
value: DstrategyStruct = driver.instrument.select.dstrategy.get()
```

No command help available

return
structure: for return value, see the help for DstrategyStruct structure arguments.

set(*arg_0: OperationMode*, *arg_1: Dstrategy = None*) → None

```
# SCPI: INSTrument[:SELect]:DSTRategy
driver.instrument.select.dstrategy.set(arg_0 = enums.OperationMode.LOCal, arg_1␣
↪= enums.Dstrategy.BYLayout)
```

No command help available

param arg_0
No help available

param arg_1
No help available

6.11 MassMemory

SCPI Command:

```
MMEMory:COPY
MMEMory:DELete
MMEMory:DRIVes
MMEMory:MDIRectory
MMEMory:MOVE
MMEMory:MSIS
MMEMory:RDIRectory
```

class MassMemoryCls

MassMemory commands group definition. 17 total commands, 6 Subgroups, 7 group commands

copy(*file_source: str*, *file_destination: str = None*) → None

```
# SCPI: MMEMory:COPY
driver.massMemory.copy(file_source = '1', file_destination = '1')
```

Copies an existing file. The target directory must exist.

param file_source
String parameter to specify the name of the file to be copied. Wildcards ? and * are allowed if FileDestination contains a path without file name.

param file_destination
String parameter to specify the path and/or name of the new file. If the parameter is omitted, the new file is written to the current directory (see method RsCma.MassMemory.CurrentDirectory.set) .

delete(*filename: str*) → None

```
# SCPI: MMEMory:DELete
driver.massMemory.delete(filename = '1')
```

Deletes the specified files.

param filename
String parameter to specify the file to be deleted. The wildcards * and ? are allowed. Specifying a directory instead of a file is not allowed.

delete_directory(*directory_name: str*) → None

```
# SCPI: MMEMory:RDIRectory
driver.massMemory.delete_directory(directory_name = '1')
```

Deletes an existing empty directory.

param directory_name
String parameter to specify the directory.

get_drives() → List[str]

```
# SCPI: MMEMory:DRIVes
value: List[str] = driver.massMemory.get_drives()
```

Returns a list of the drives of the instrument.

return
drive: No help available

get_msis() → str

```
# SCPI: MMEMory:MSIS
value: str = driver.massMemory.get_msis()
```

Sets the default storage unit to the specified drive or network server. When the default storage unit is changed, the CMA checks whether the current directory (see method RsCma.MassMemory.CurrentDirectory.set) is also available on the new storage unit. If not, the current directory is automatically set to '/'.

return
msus: No help available

make_directory(*directory_name: str*) → None

```
# SCPI: MMEMory:MDIRectory
driver.massMemory.make_directory(directory_name = '1')
```

Creates a directory.

param directory_name
String parameter to specify the new directory. All not yet existing parts of the specified path are created.

move(*file_source: str*, *file_destination: str*) → None

```
# SCPI: MMEMory:MOVE
driver.massMemory.move(file_source = '1', file_destination = '1')
```

Moves an existing object (file or directory) to a new location and renames it.

param file_source
String parameter to specify the name of the object to be moved or renamed. Wildcards ? and * are allowed if the files are not renamed.

param file_destination
String parameter to specify the new name and/or path of the object. New object name without path: The object is renamed. New path without object name: The object is moved. New path and new object name: The object is moved and renamed.

set_msis(*msus: str*) → None

```
# SCPI: MMEMory:MSIS
driver.massMemory.set_msis(msus = '1')
```

Sets the default storage unit to the specified drive or network server. When the default storage unit is changed, the CMA checks whether the current directory (see method RsCma.MassMemory.CurrentDirectory.set) is also available on the new storage unit. If not, the current directory is automatically set to '/'.

param msus
String parameter to specify the default storage unit. If the parameter is omitted, the storage unit is set to D:.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.massMemory.clone()
```

Subgroups

6.11.1 Attribute

SCPI Command:

```
MMEMory:ATTRibute
```

class AttributeCls

Attribute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*path_name: str*) → List[str]

```
# SCPI: MMEMory:ATTRibute
value: List[str] = driver.massMemory.attribute.get(path_name = '1')
```

Sets or removes attributes for files and directories, or queries the attributes.

param path_name
No help available

return
file_entry: No help available

set(*path_name: str*, *attributes: str*) → None

```
# SCPI: MMEMory:ATTRibute
driver.massMemory.attribute.set(path_name = '1', attributes = '1')
```

Sets or removes attributes for files and directories, or queries the attributes.

param path_name
No help available

param attributes
Attribute actions, separated by a blank + before an attribute: Sets the attribute. - before an attribute: Deletes the attribute. Possible attributes: R: read-only file A: archive file S: system file H: hidden file

6.11.2 Catalog

SCPI Command:

```
MMEMory:CATalog
```

class CatalogCls

Catalog commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Used_Memory: int: No parameter help available
- Free_Memory: int: No parameter help available
- File_Entry: List[str]: No parameter help available

get(*path_name: str*, *format_py: CatalogFormat = None*) → GetStruct

```
# SCPI: MMEMory:CATalog
value: GetStruct = driver.massMemory.catalog.get(path_name = '1', format_py =␣
↪enums.CatalogFormat.ALL)
```

Returns information about the specified directory.

param path_name
No help available

param format_py
ALL | WTIme ALL Output enhanced with date, time and file attributes WTIme Output enhanced with date and time

return
structure: for return value, see the help for GetStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.massMemory.catalog.clone()
```

Subgroups

6.11.2.1 Length

SCPI Command:

```
MMEMory:CATalog:LENGth
```

class LengthCls

Length commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*path_name: str = None*) → int

```
# SCPI: MMEMory:CATalog:LENGth
value: int = driver.massMemory.catalog.length.get(path_name = '1')
```

Returns the number of files and subdirectories in the specified directory. The returned number corresponds to the number of information strings returned by the method **RsCma.MassMemory.Catalog.get_** command.

param path_name
String parameter, directory to be queried. If the directory is omitted, the command queries the contents of the current directory (see method

RsCma.MassMemory.CurrentDirectory.set) . If the wildcards ? or * are used, the number of files and subdirectories matching this pattern are returned.

return
count: No help available

6.11.3 CurrentDirectory

SCPI Command:

```
MMEMory:CDIRectory
```

class CurrentDirectoryCls

CurrentDirectory commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*directory_name: str = None*) → str

```
# SCPI: MMEMory:CDIRectory
value: str = driver.massMemory.currentDirectory.get(directory_name = '1')
```

Changes the current directory for file access.

param directory_name
String parameter to specify the directory. If the parameter is omitted, the current directory is set to '/'. If the string contains not only a directory, but also a drive letter or server name, the command MMEMory:MSIS is also executed automatically.

return
directory_name: String parameter to specify the directory. If the parameter is omitted, the current directory is set to '/'. If the string contains not only a directory, but also a drive letter or server name, the command MMEMory:MSIS is also executed automatically.

set(*directory_name: str = None*) → None

```
# SCPI: MMEMory:CDIRectory
driver.massMemory.currentDirectory.set(directory_name = '1')
```

Changes the current directory for file access.

param directory_name
String parameter to specify the directory. If the parameter is omitted, the current directory is set to '/'. If the string contains not only a directory, but also a drive letter or server name, the command MMEMory:MSIS is also executed automatically.

6.11.4 Dcatalog

SCPI Command:

```
MMEMory:DCATalog
```

class DcatalogCls

Dcatalog commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*path_name: str = None*) → List[str]

```
# SCPI: MMEMory:DCATalog
value: List[str] = driver.massMemory.dcatalog.get(path_name = '1')
```

Returns the subdirectories of the specified directory.

param path_name
No help available

return
file_entry: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.massMemory.dcatalog.clone()
```

Subgroups

6.11.4.1 Length

SCPI Command:

```
MMEMory:DCATalog:LENGth
```

class LengthCls

Length commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*path_name: str = None*) → int

```
# SCPI: MMEMory:DCATalog:LENGth
value: int = driver.massMemory.dcatalog.length.get(path_name = '1')
```

Returns the number of subdirectories of the specified directory. The number corresponds to the number of strings returned by the method **RsCma.MassMemory.Dcatalog.get_** command.

param path_name
String parameter to specify the directory. If the parameter is omitted, the command queries the contents of the current directory (see method RsCma.MassMemory.CurrentDirectory.set) . If the wildcards ? or * are used, the number of subdirectories matching this pattern are returned.

return
file_entry_count: No help available

6.11.5 Load

class LoadCls

Load 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.massMemory.load.clone()
```

Subgroups

6.11.5.1 Item

SCPI Command:

```
MMEMory:LOAD:ITEM
```

class ItemCls

Item commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*item_path: str*, *filename: str*) → None

```
# SCPI: MMEMory:LOAD:ITEM
driver.massMemory.load.item.set(item_path = '1', filename = '1')
```

No command help available

param item_path
No help available

param filename
No help available

6.11.5.2 State

SCPI Command:

```
MMEMory:LOAD:STATe
```

class StateCls

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*sav_rcl_state_number: float*, *filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:LOAD:STATe
driver.massMemory.load.state.set(sav_rcl_state_number = 1.0, filename = '1',␣
↪msus = '1')
```

Loads the instrument settings from the specified file to the specified internal memory. After the file has been loaded, the settings must be activated using a *RCL command.

param sav_rcl_state_number
No help available

param filename
No help available

param msus
No help available

6.11.6 Store

class StoreCls

Store 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.massMemory.store.clone()
```

Subgroups

6.11.6.1 Item

SCPI Command:

```
MMEMory:STORe:ITEM
```

class ItemCls

Item commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*item_name: str*, *filename: str*) → None

```
# SCPI: MMEMory:STORe:ITEM
driver.massMemory.store.item.set(item_name = '1', filename = '1')
```

No command help available

param item_name
No help available

param filename
No help available

6.11.6.2 State

SCPI Command:

```
MMEMory:STORe:STATe
```

class StateCls

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*sav_rcl_state_number: int*, *filename: str*, *msus: str = None*) → None

```
# SCPI: MMEMory:STORe:STATe
driver.massMemory.store.state.set(sav_rcl_state_number = 1, filename = '1',␣
↪msus = '1')
```

Stores the instrument settings from the specified internal memory to the specified file. To store the current instrument settings to the memory, use *SAV <MemoryNumber> first.

param sav_rcl_state_number
No help available

param filename
No help available

param msus
No help available

6.12 RecallState

SCPI Command:

```
*RCL
```

class RecallStateCls

RecallState commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*num: float*) → None

```
# SCPI: *RCL
driver.recallState.set(num = 1.0)
```

Loads the instrument settings from an intermediate memory identified by the specified number. The instrument settings can be stored to this memory using the command *SAV with the associated number. To load instrument settings from a file to the memory, see method RsCma.MassMemory.Load.State.set. See also MMEMory:RCL.

param num
Range: 0 to 99

6.13 SaveState

SCPI Command:

```
*SAV
```

class SaveStateCls

SaveState commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*num: float*) → None

```
# SCPI: *SAV
driver.saveState.set(num = 1.0)
```

Stores the current instrument settings under the specified number in an intermediate memory. The settings can be restored, using the command *RCL with the associated number. To save the stored instrument settings to a file, see method RsCma. MassMemory.Store.State.set. See also MMEMory:SAV.

param num
Range: 0 to 99

6.14 Sense

class SenseCls

Sense commands group definition. 17 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.clone()
```

Subgroups

6.14.1 Base

class BaseCls

Base commands group definition. 11 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.base.clone()
```

Subgroups

6.14.1.1 Battery<Battery>

RepCap Settings

```
# Range: Ix1 .. Ix2
rc = driver.sense.base.battery.repcap_battery_get()
driver.sense.base.battery.repcap_battery_set(repcap.Battery.Ix1)
```

SCPI Command:

```
SENSe:BASE:BATTery:AVAilable
SENSe:BASE:BATTery:CAPacity
SENSe:BASE:BATTery:TTD
SENSe:BASE:BATTery:USAGe
```

class BatteryCls

Battery commands group definition. 5 total commands, 1 Subgroups, 4 group commands Repeated Capability: Battery, default value after init: Battery.Ix1

get_available() → bool

```
# SCPI: SENSe:BASE:BATTery:AVAilable
value: bool = driver.sense.base.battery.get_available()
```

Queries if at least one battery is inserted.

return
batt_available: OFF | ON OFF: No battery inserted ON: One or two batteries inserted

get_capacity() → float

```
# SCPI: SENSe:BASE:BATTery:CAPacity
value: float = driver.sense.base.battery.get_capacity()
```

Queries the total capacity (sum of available batteries) .

return
batt_capacity: Range: 0 % to 100 %, Unit: %

get_ttd() → int

```
# SCPI: SENSe:BASE:BATTery:TTD
value: int = driver.sense.base.battery.get_ttd()
```

Queries the estimated total remaining runtime for the sum of all available batteries. The value is calculated from the total capacity and the current discharge rate of the used battery.

return
ttd: Time until discharged Unit: s

get_usage() → List[BatteryUsage]

```
# SCPI: SENSe:BASE:BATTery:USAGe
value: List[enums.BatteryUsage] = driver.sense.base.battery.get_usage()
```

Queries the state of both slots of the battery compartment. Two values are returned: <BattUsage>slot 1, <BattUsage>slot 2

return
batt_usage: NAV | REMovable | USED NAV Slot empty REMovable Battery inserted but currently not used - can be removed USED Battery currently used - do not remove it

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.base.battery.clone()
```

Subgroups

6.14.1.1.1 Info

SCPI Command:

```
SENSe:BASE:BATTery<BattIdx>:INFO
```

class InfoCls

Info commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Usage: enums.BatteryUsage: NAV | REMovable | USED NAV Slot empty REMovable Battery inserted but currently not used - can be removed USED Battery currently used - do not remove it
- Capacity: float: Battery capacity Unit: %
- Design_Cap: float: Nominal capacity stated by the battery manufacturer Unit: Wh
- Full_Ch_Cap: float: Full-charge capacity of the battery Unit: Wh
- Voltage: float: Battery voltage Unit: V
- Temp: float: Battery temperature Unit: Deg
- Disch_Rate: float: Discharge rate Unit: W
- Cycle_Count: int: Charge/discharge cycles
- Dev_Name: str: Battery name as string
- Serial_Nr: str: Battery serial number as string
- Manufacturer_Name: str: Battery manufacturer as string
- Manufacture_Date: str: Battery manufacturing date as string

get(*battery=Battery.Default*) → GetStruct

```
# SCPI: SENSe:BASE:BATTery<BattIdx>:INFO
value: GetStruct = driver.sense.base.battery.info.get(battery = repcap.Battery.
↪Default)
```

Queries information for a battery slot.

param battery
optional repeated capability selector. Default value: Ix1 (settable in the interface 'Battery')

return
structure: for return value, see the help for GetStruct structure arguments.

6.14.1.2 Power

SCPI Command:

```
SENSe:BASE:POWer:OMODe
```

class PowerCls

Power commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_omode() → SupplyMode

```
# SCPI: SENSe:BASE:POWer:OMODe
value: enums.SupplyMode = driver.sense.base.power.get_omode()
```

Queries whether the instrument is powered by an inserted battery or by an external power supply.

return
oper_mode: BATTery | MAINs

6.14.1.3 Reference

class ReferenceCls

Reference commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.base.reference.clone()
```

Subgroups

6.14.1.3.1 Frequency

SCPI Command:

```
SENSe:BASE:REFerence:FREQuency:LOCKed
SENSe:BASE:REFerence:FREQuency:OVENcold
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_locked() → bool

```
# SCPI: SENSe:BASE:REFerence:FREQuency:LOCKed
value: bool = driver.sense.base.reference.frequency.get_locked()
```

Queries whether the reference frequency is locked or not.

return
lock: 1 | 0 1: Frequency is locked 0: Frequency is not locked

get_oven_cold() → bool

```
# SCPI: SENSe:BASE:REFerence:FREQuency:OVENcold
value: bool = driver.sense.base.reference.frequency.get_oven_cold()
```

Queries whether an installed OCXO has completed the warm-up phase and has reached its operating temperature.

return
oven_cold: 0 | 1 0: warm-up completed, operating temperature reached 1: oven still cold, warm-up is ongoing

6.14.1.4 Temperature

SCPI Command:

```
SENSe:BASE:TEMPerature:ENVironment
```

class TemperatureCls

Temperature commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_environment() → float

```
# SCPI: SENSe:BASE:TEMPerature:ENVironment
value: float = driver.sense.base.temperature.get_environment()
```

No command help available

return
temperature: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.base.temperature.clone()
```

Subgroups

6.14.1.4.1 Exceeded

SCPI Command:

```
SENSe:BASE:TEMPerature:EXCeeded:LIST
SENSe:BASE:TEMPerature:EXCeeded
```

class ExceededCls

Exceeded commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ListPyStruct

Structure for reading output parameters. Fields:

- Meas_Point: List[str]: No parameter help available
- Current_Temp: List[float]: No parameter help available
- Max_Temp: List[float]: No parameter help available

get_list_py() → ListPyStruct

```
# SCPI: SENSe:BASE:TEMPerature:EXCeeded:LIST
value: ListPyStruct = driver.sense.base.temperature.exceeded.get_list_py()
```

No command help available

return
structure: for return value, see the help for ListPyStruct structure arguments.

get_value() → bool

```
# SCPI: SENSe:BASE:TEMPerature:EXCeeded
value: bool = driver.sense.base.temperature.exceeded.get_value()
```

No command help available

return
exceed: No help available

6.14.2 Display

class DisplayCls

Display commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.display.clone()
```

Subgroups

6.14.2.1 Applications

SCPI Command:

```
SENSe:DISPlay:APPLications:CATalog
```

class ApplicationsCls

Applications commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_catalog() → List[str]

```
# SCPI: SENSe:DISPlay:APPLications:CATalog
value: List[str] = driver.sense.display.applications.get_catalog()
```

Queries a list of all applications available in the current scenario.

return
app_list: Comma-separated list of strings, each string indicating one application, for example 'Generator' or 'Analyzer'

6.14.3 FirmwareUpdate

SCPI Command:

```
SENSe:FWUPdate:INFO
```

class FirmwareUpdateCls

FirmwareUpdate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_info() → str

```
# SCPI: SENSe:FWUPdate:INFO
value: str = driver.sense.firmwareUpdate.get_info()
```

No command help available

return
info: No help available

6.14.4 Sequencer

class SequencerCls

Sequencer commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.sequencer.clone()
```

Subgroups

6.14.4.1 Tplan

SCPI Command:

```
SENSe:SEQuencer:TPLan:LIST
SENSe:SEQuencer:TPLan:INFO
```

class TplanCls

Tplan commands group definition. 4 total commands, 2 Subgroups, 2 group commands

class InfoStruct

Structure for reading output parameters. Fields:

- Tp_Name: List[str]: No parameter help available
- State: List[enums.TestPlanState]: No parameter help available
- Status: List[enums.Status]: No parameter help available

get_info() → InfoStruct

```
# SCPI: SENSe:SEQuencer:TPLan:INFO
value: InfoStruct = driver.sense.sequencer.tplan.get_info()
```

No command help available

return
structure: for return value, see the help for InfoStruct structure arguments.

get_list_py() → List[str]

```
# SCPI: SENSe:SEQuencer:TPLan:LIST
value: List[str] = driver.sense.sequencer.tplan.get_list_py()
```

No command help available

return
tp_list: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.sequencer.tplan.clone()
```

Subgroups

6.14.4.1.1 Estatus

SCPI Command:

```
SENSe:SEQuencer:TPLan:ESTatus
```

class EstatusCls

Estatus commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*tp_name: str*) → Status

```
# SCPI: SENSe:SEQuencer:TPLan:ESTatus
value: enums.Status = driver.sense.sequencer.tplan.estatus.get(tp_name = '1')
```

No command help available

param tp_name
No help available

return
status: No help available

6.14.4.1.2 State

SCPI Command:

```
SENSe:SEQuencer:TPLan:STATe
```

class StateCls

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*tp_name: str*) → TestPlanState

```
# SCPI: SENSe:SEQuencer:TPLan:STATe
value: enums.TestPlanState = driver.sense.sequencer.tplan.state.get(tp_name = '1
↪')
```

No command help available

param tp_name
No help available

return
state: No help available

6.15 Source

class SourceCls

Source commands group definition. 339 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.clone()
```

Subgroups

6.15.1 AfRf

class AfRfCls

AfRf commands group definition. 240 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.clone()
```

Subgroups

6.15.1.1 Generator

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DSOurce
SOURce:AFRF:GENerator<Instance>:MSCHeme
```

class GeneratorCls

Generator commands group definition. 240 total commands, 23 Subgroups, 2 group commands

get_dsource() → DigitalSource

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DSOurce
value: enums.DigitalSource = driver.source.afRf.generator.get_dsource()
```

Selects the data source for digital scenarios.

return
dsource: DMR | ARB | NXDN | POCSag | P25 | UDEFined | ZIGBee | DPMR

get_mscheme() → ModulationScheme

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MSCHeme
value: enums.ModulationScheme = driver.source.afRf.generator.get_mscheme()
```

Selects the RF signal mode (modulation scheme) for analog scenarios.

return
mod_scheme: FMSTereo | FM | AM | USB | LSB | PM | CW | ARB FMSTereo FM stereo multiplex signal FM, PM, AM Frequency / phase / amplitude modulation USB, LSB Single sideband modulation, upper / lower sideband CW Constant wave signal (unmodulated RF carrier) ARB Waveform file (ARB file)

set_dsource(*dsource: DigitalSource*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DSOurce
driver.source.afRf.generator.set_dsource(dsource = enums.DigitalSource.ARB)
```

Selects the data source for digital scenarios.

param dsource
DMR | ARB | NXDN | POCSag | P25 | UDEFined | ZIGBee | DPMR

set_mscheme(*mod_scheme: ModulationScheme*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MSCHeme
driver.source.afRf.generator.set_mscheme(mod_scheme = enums.ModulationScheme.AM)
```

Selects the RF signal mode (modulation scheme) for analog scenarios.

param mod_scheme
FMSTereo | FM | AM | USB | LSB | PM | CW | ARB FMSTereo FM stereo multiplex signal FM, PM, AM Frequency / phase / amplitude modulation USB, LSB Single sideband modulation, upper / lower sideband CW Constant wave signal (unmodulated RF carrier) ARB Waveform file (ARB file)

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.clone()
```

Subgroups

6.15.1.1.1 Arb

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:ARB:CRATe
SOURce:AFRF:GENerator<Instance>:ARB:CRCProtect
SOURce:AFRF:GENerator<Instance>:ARB:FOFFset
SOURce:AFRF:GENerator<Instance>:ARB:LOFFset
SOURce:AFRF:GENerator<Instance>:ARB:POFFset
SOURce:AFRF:GENerator<Instance>:ARB:REPetition
```

class ArbCls

Arb commands group definition. 13 total commands, 3 Subgroups, 6 group commands

get_crate() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:CRATe
value: float = driver.source.afRf.generator.arb.get_crate()
```

Queries the clock rate of the loaded ARB file.

return
clock_rate: Unit: Hz

get_crc_protect() → YesNoStatus

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:CRCProtect
value: enums.YesNoStatus = driver.source.afRf.generator.arb.get_crc_protect()
```

Queries whether the loaded ARB file contains a CRC checksum.

return
crc_protection: NO | YES

get_foffset() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:FOFFset
value: float = driver.source.afRf.generator.arb.get_foffset()
```

Defines a frequency offset to be imposed at the baseband during ARB generation.

return
frequency_offset: Range: -10 MHz to 10 MHz, Unit: Hz

get_loffset() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:LOFFset
value: float = driver.source.afRf.generator.arb.get_loffset()
```

Queries the peak to average ratio (PAR) of the loaded ARB file. The PAR is also called level offset.

return
level_offset: Unit: dB

get_poffset() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:POFFset
value: float = driver.source.afRf.generator.arb.get_poffset()
```

Queries the peak offset of the loaded ARB file.

return
peak_offset: Unit: dB

get_repetition() → RepeatMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:REPetition
value: enums.RepeatMode = driver.source.afRf.generator.arb.get_repetition()
```

Defines how often the ARB file is processed.

return
repetition: CONTinuous | SINGle CONTinuous Cyclic continuous processing SINGle File is processed once

set_foffset(*frequency_offset: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:FOFFset
driver.source.afRf.generator.arb.set_foffset(frequency_offset = 1.0)
```

Defines a frequency offset to be imposed at the baseband during ARB generation.

param frequency_offset
Range: -10 MHz to 10 MHz, Unit: Hz

set_repetition(*repetition: RepeatMode*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:REPetition
driver.source.afRf.generator.arb.set_repetition(repetition = enums.RepeatMode.
↪CONTinuous)
```

Defines how often the ARB file is processed.

param repetition
CONTinuous | SINGle CONTinuous Cyclic continuous processing SINGle File is processed once

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.arb.clone()
```

Subgroups

6.15.1.1.1.1 File

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:ARB:FILE:DATE
SOURce:AFRF:GENerator<Instance>:ARB:FILE:OPTion
SOURce:AFRF:GENerator<Instance>:ARB:FILE:VERSion
SOURce:AFRF:GENerator<Instance>:ARB:FILE
```

class FileCls

File commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_date() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:FILE:DATE
value: str = driver.source.afRf.generator.arb.file.get_date()
```

Queries the date and time of the loaded ARB file.

return
date: String with date and time

get_option() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:FILE:OPTion
value: str = driver.source.afRf.generator.arb.file.get_option()
```

Queries the options that are required to process the loaded ARB file.

return
options: String with comma-separated list of options

get_value() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:FILE
value: str = driver.source.afRf.generator.arb.file.get_value()
```

Selects the ARB file to be processed. Specify the path and the filename. If the file is stored in the folder corresponding to the @waveform alias, it is sufficient to specify only the filename.

INTRO_CMD_HELP: Example, the following strings are equivalent:

- 'D:/Rohde-Schwarz/CMA/Data/waveform/myfile.wv'
- '@WAVEFORM/myfile.wv'
- 'myfile.wv'

return
arb_file: String specifying the ARB file

get_version() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:FILE:VERSion
value: str = driver.source.afRf.generator.arb.file.get_version()
```

Queries the version of the loaded ARB file.

return
version: String containing the version Empty string, if no file version is defined

set_value(*arb_file: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:FILE
driver.source.afRf.generator.arb.file.set_value(arb_file = '1')
```

Selects the ARB file to be processed. Specify the path and the filename. If the file is stored in the folder corresponding to the @waveform alias, it is sufficient to specify only the filename.

INTRO_CMD_HELP: Example, the following strings are equivalent:

- 'D:/Rohde-Schwarz/CMA/Data/waveform/myfile.wv'
- '@WAVEFORM/myfile.wv'
- 'myfile.wv'

param arb_file
String specifying the ARB file

6.15.1.1.1.2 Marker

class MarkerCls

Marker commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.arb.marker.clone()
```

Subgroups

6.15.1.1.1.3 Delays

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:ARB:MARKer:DELays
```

class DelaysCls

Delays commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DelaysStruct

Response structure. Fields:

- Marker_2: int: Delay for marker 2 of the ARB file Range: -10 to 4000
- Marker_3: int: Delay for marker 3 of the ARB file Range: -10 to 4000
- Marker_4: int: Delay for marker 4 of the ARB file Range: -10 to 4000
- Restart_Marker: int: Delay for marker event due to start of ARB file processing Range: 0 to n (depends on ARB file)

get() → DelaysStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:MARKer:DELays
value: DelaysStruct = driver.source.afRf.generator.arb.marker.delays.get()
```

Defines delay times for the generation of trigger signals relative to the marker events. All delay times are specified as number of samples.

return
structure: for return value, see the help for DelaysStruct structure arguments.

set(*marker_2: int*, *marker_3: int*, *marker_4: int*, *restart_marker: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:MARKer:DELays
driver.source.afRf.generator.arb.marker.delays.set(marker_2 = 1, marker_3 = 1,␣
↪marker_4 = 1, restart_marker = 1)
```

Defines delay times for the generation of trigger signals relative to the marker events. All delay times are specified as number of samples.

param marker_2
Delay for marker 2 of the ARB file Range: -10 to 4000

param marker_3
Delay for marker 3 of the ARB file Range: -10 to 4000

param marker_4
Delay for marker 4 of the ARB file Range: -10 to 4000

param restart_marker
Delay for marker event due to start of ARB file processing Range: 0 to n (depends on ARB file)

6.15.1.1.1.4 Samples

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:ARB:SAMPles
```

class SamplesCls

Samples commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:SAMPles
value: float = driver.source.afRf.generator.arb.samples.get_value()
```

Queries the number of samples in the loaded ARB file.

return
samples: Range: 0 to 268173312

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.arb.samples.clone()
```

Subgroups

6.15.1.1.1.5 Range

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:ARB:SAMPles:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RangeStruct

Response structure. Fields:

- Range_Py: enums.ArbSamplesRange: FULL | SUB FULL Process all samples SUB Process a subrange according to Start and Stop
- Start: int: Start of the subrange (always first sample, labeled zero) Range: 0 (fixed value)

- Stop: int: End of the subrange Range: 16 to samples in ARB file - 1

get() → RangeStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:SAMPles:RANGe
value: RangeStruct = driver.source.afRf.generator.arb.samples.range.get()
```

Selects whether all samples or a subrange of samples is processed.

return
structure: for return value, see the help for RangeStruct structure arguments.

set(*range_py: ArbSamplesRange*, *start: int = None*, *stop: int = None*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ARB:SAMPles:RANGe
driver.source.afRf.generator.arb.samples.range.set(range_py = enums.
↪ArbSamplesRange.FULL, start = 1, stop = 1)
```

Selects whether all samples or a subrange of samples is processed.

param range_py
FULL | SUB FULL Process all samples SUB Process a subrange according to Start and Stop

param start
Start of the subrange (always first sample, labeled zero) Range: 0 (fixed value)

param stop
End of the subrange Range: 16 to samples in ARB file - 1

6.15.1.1.2 AudioInput<AudioInput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.source.afRf.generator.audioInput.repcap_audioInput_get()
driver.source.afRf.generator.audioInput.repcap_audioInput_set(repcap.AudioInput.Nr1)
```

class AudioInputCls

AudioInput commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: AudioInput, default value after init: AudioInput.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.audioInput.clone()
```

Subgroups

6.15.1.1.2.1 Aranging

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AIN<nr>:ARANging
```

class ArangingCls

Aranging commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN<nr>:ARANging
value: bool = driver.source.afRf.generator.audioInput.aranging.get(audioInput =␣
↪repcap.AudioInput.Default)
```

Enables or disables auto ranging for an AF IN connector.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
enable: OFF | ON Switches auto ranging off or on

set(*enable: bool*, *audioInput=AudioInput.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN<nr>:ARANging
driver.source.afRf.generator.audioInput.aranging.set(enable = False, audioInput␣
↪= repcap.AudioInput.Default)
```

Enables or disables auto ranging for an AF IN connector.

param enable
OFF | ON Switches auto ranging off or on

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.15.1.1.2.2 First

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AIN:FIRSt:MLEVel
```

class FirstCls

First commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mlevel() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN:FIRSt:MLEVel
value: float = driver.source.afRf.generator.audioInput.first.get_mlevel()
```

Specifies the maximum expected level for the AF1 IN connector. This setting is only relevant, if auto ranging is disabled.

return
level: Maximum expected level Range: 10E-6 V to 43 V, Unit: V

set_mlevel(*level: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN:FIRSt:MLEVel
driver.source.afRf.generator.audioInput.first.set_mlevel(level = 1.0)
```

Specifies the maximum expected level for the AF1 IN connector. This setting is only relevant, if auto ranging is disabled.

param level
Maximum expected level Range: 10E-6 V to 43 V, Unit: V

6.15.1.1.2.3 Icoupling

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AIN<nr>:ICOupling
```

class IcouplingCls

Icoupling commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → PathCoupling

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN<nr>:ICOupling
value: enums.PathCoupling = driver.source.afRf.generator.audioInput.icoupling.
↪get(audioInput = repcap.AudioInput.Default)
```

Configures whether the DC signal component is blocked at an AF IN connector, or not.

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
input_coupling: AC | DC AC DC component blocked, only AC component available DC AC and DC component available

set(*input_coupling: PathCoupling*, *audioInput=AudioInput.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN<nr>:ICOupling
driver.source.afRf.generator.audioInput.icoupling.set(input_coupling = enums.
↪PathCoupling.AC, audioInput = repcap.AudioInput.Default)
```

Configures whether the DC signal component is blocked at an AF IN connector, or not.

param input_coupling
AC | DC AC DC component blocked, only AC component available DC AC and DC component available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.15.1.1.2.4 Mlevel

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AIN<nr>:MLEVel
```

class MlevelCls

Mlevel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioInput=AudioInput.Default*) → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN<nr>:MLEVel
value: float = driver.source.afRf.generator.audioInput.mlevel.get(audioInput =␣
↪repcap.AudioInput.Default)
```

No command help available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

return
level: No help available

set(*level: float*, *audioInput=AudioInput.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN<nr>:MLEVel
driver.source.afRf.generator.audioInput.mlevel.set(level = 1.0, audioInput =␣
↪repcap.AudioInput.Default)
```

No command help available

param level
No help available

param audioInput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioInput')

6.15.1.1.2.5 Second

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AIN:SECond:MLEVel
```

class SecondCls

Second commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mlevel() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN:SECond:MLEVel
value: float = driver.source.afRf.generator.audioInput.second.get_mlevel()
```

Specifies the maximum expected level for the AF2 IN connector. This setting is only relevant, if auto ranging is disabled.

return
level: Maximum expected level Range: 10E-6 V to 43 V, Unit: V

set_mlevel(*level: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AIN:SECond:MLEVel
driver.source.afRf.generator.audioInput.second.set_mlevel(level = 1.0)
```

Specifies the maximum expected level for the AF2 IN connector. This setting is only relevant, if auto ranging is disabled.

param level
Maximum expected level Range: 10E-6 V to 43 V, Unit: V

6.15.1.1.3 AudioOutput<AudioOutput>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.source.afRf.generator.audioOutput.repcap_audioOutput_get()
driver.source.afRf.generator.audioOutput.repcap_audioOutput_set(repcap.AudioOutput.Nr1)
```

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AOUT<nr>
```

class AudioOutputCls

AudioOutput commands group definition. 5 total commands, 4 Subgroups, 1 group commands Repeated Capability: AudioOutput, default value after init: AudioOutput.Nr1

get(*audioOutput=AudioOutput.Default*) → SignalSource

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT<nr>
value: enums.SignalSource = driver.source.afRf.generator.audioOutput.
↪get(audioOutput = repcap.AudioOutput.Default)
```

Selects an audio signal source for an AF OUT connector.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
af_out_source: GEN1 | GEN2 | AFI1 | AFI2 | SPIL | SPIR GEN1 AF1 OUT source: Audio generator 1 AFI1 AF1 OUT source: AF1 IN SPIL AF1 OUT source: SPDIF IN, left channel GEN2 AF2 OUT source: Audio generator 2 AFI2 AF2 OUT source: AF2 IN SPIR AF2 OUT source: SPDIF IN, right channel

set(*af_out_source: SignalSource*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT<nr>
driver.source.afRf.generator.audioOutput.set(af_out_source = enums.SignalSource.
↪AFI1, audioOutput = repcap.AudioOutput.Default)
```

Selects an audio signal source for an AF OUT connector.

param af_out_source
GEN1 | GEN2 | AFI1 | AFI2 | SPIL | SPIR GEN1 AF1 OUT source: Audio generator 1 AFI1 AF1 OUT source: AF1 IN SPIL AF1 OUT source: SPDIF IN, left channel GEN2 AF2 OUT source: Audio generator 2 AFI2 AF2 OUT source: AF2 IN SPIR AF2 OUT source: SPDIF IN, right channel

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.audioOutput.clone()
```

Subgroups

6.15.1.1.3.1 Enable

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AOUT<nr>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioOutput=AudioOutput.Default*) → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT<nr>:ENABle
value: bool = driver.source.afRf.generator.audioOutput.enable.get(audioOutput =␣
↪repcap.AudioOutput.Default)
```

Enables or disables an AF OUT connector.

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
enable: OFF | ON

set(*enable: bool*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT<nr>:ENABle
driver.source.afRf.generator.audioOutput.enable.set(enable = False, audioOutput␣
↪= repcap.AudioOutput.Default)
```

Enables or disables an AF OUT connector.

param enable
OFF | ON

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.15.1.1.3.2 First

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AOUT:FIRSt:LEVel
```

class FirstCls

First commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_level() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT:FIRSt:LEVel
value: float = driver.source.afRf.generator.audioOutput.first.get_level()
```

Specifies the output level for the AF1 OUT connector. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

return
level: Output level Range: 10E-6 V to 5 V, Unit: V

set_level(*level: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT:FIRSt:LEVel
driver.source.afRf.generator.audioOutput.first.set_level(level = 1.0)
```

Specifies the output level for the AF1 OUT connector. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

param level
Output level Range: 10E-6 V to 5 V, Unit: V

6.15.1.1.3.3 Level

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AOUT<nr>:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*audioOutput=AudioOutput.Default*) → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT<nr>:LEVel
value: float = driver.source.afRf.generator.audioOutput.level.get(audioOutput =␣
↪repcap.AudioOutput.Default)
```

No command help available

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

return
level: No help available

set(*level: float*, *audioOutput=AudioOutput.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT<nr>:LEVel
driver.source.afRf.generator.audioOutput.level.set(level = 1.0, audioOutput =␣
↪repcap.AudioOutput.Default)
```

No command help available

param level
No help available

param audioOutput
optional repeated capability selector. Default value: Nr1 (settable in the interface 'AudioOutput')

6.15.1.1.3.4 Second

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:AOUT:SECond:LEVel
```

class SecondCls

Second commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_level() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT:SECond:LEVel
value: float = driver.source.afRf.generator.audioOutput.second.get_level()
```

Specifies the output level for the AF2 OUT connector. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

return
level: Output level Range: 10E-6 V to 5 V, Unit: V

set_level(*level: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:AOUT:SECond:LEVel
driver.source.afRf.generator.audioOutput.second.set_level(level = 1.0)
```

Specifies the output level for the AF2 OUT connector. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

param level
Output level Range: 10E-6 V to 5 V, Unit: V

6.15.1.1.4 Cdefinition

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:CDEFinition:RCHannel
SOURce:AFRF:GENerator<Instance>:CDEFinition:RFRequency
SOURce:AFRF:GENerator<Instance>:CDEFinition:CSPace
SOURce:AFRF:GENerator<Instance>:CDEFinition
```

class CdefinitionCls

Cdefinition commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_cspace() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition:CSPace
value: float = driver.source.afRf.generator.cdefinition.get_cspace()
```

Defines the channel spacing, that is the center frequency difference of two adjacent channels. This setting is part of the channel definition.

return
channel_space: Range: 100 Hz to 4 MHz, Unit: Hz

get_rchannel() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition:RCHannel
value: int = driver.source.afRf.generator.cdefinition.get_rchannel()
```

Assigns a reference channel number to the reference frequency defined via method RsCma.Source.AfRf.Generator.Cdefinition. rfrequency. This setting is part of the channel definition.

return
reference_ch: Range: 0 Ch to 9999 Ch, Unit: Ch

get_rfrequency() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition:RFRequency
value: float = driver.source.afRf.generator.cdefinition.get_rfrequency()
```

Assigns a reference frequency to the reference channel number defined via method RsCma.Source.AfRf.Generator.Cdefinition. rchannel. This setting is part of the channel definition.

return
reference_freq: Range: 100 kHz to 3 GHz, Unit: Hz

get_value() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition
value: bool = driver.source.afRf.generator.cdefinition.get_value()
```

Activates or deactivates the channel definition.

return
enable: OFF | ON

set_cspace(*channel_space: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition:CSPace
driver.source.afRf.generator.cdefinition.set_cspace(channel_space = 1.0)
```

Defines the channel spacing, that is the center frequency difference of two adjacent channels. This setting is part of the channel definition.

param channel_space
Range: 100 Hz to 4 MHz, Unit: Hz

set_rchannel(*reference_ch: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition:RCHannel
driver.source.afRf.generator.cdefinition.set_rchannel(reference_ch = 1)
```

Assigns a reference channel number to the reference frequency defined via method RsCma.Source.AfRf.Generator.Cdefinition. rfrequency. This setting is part of the channel definition.

param reference_ch
Range: 0 Ch to 9999 Ch, Unit: Ch

set_rfrequency(*reference_freq: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition:RFRequency
driver.source.afRf.generator.cdefinition.set_rfrequency(reference_freq = 1.0)
```

Assigns a reference frequency to the reference channel number defined via method RsCma.Source.AfRf.Generator.Cdefinition. rchannel. This setting is part of the channel definition.

param reference_freq
Range: 100 kHz to 3 GHz, Unit: Hz

set_value(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:CDEFinition
driver.source.afRf.generator.cdefinition.set_value(enable = False)
```

Activates or deactivates the channel definition.

param enable
OFF | ON

6.15.1.1.5 Dialing

class DialingCls

Dialing commands group definition. 38 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.dialing.clone()
```

Subgroups

6.15.1.1.5.1 Dtmf

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SEQuence
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SREPeat
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SPAuse
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:DTIMe
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:DPAuse
```

class DtmfCls

Dtmf commands group definition. 8 total commands, 2 Subgroups, 5 group commands

get_dpause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:DPAuse
value: float = driver.source.afRf.generator.dialing.dtmf.get_dpause()
```

Defines the duration of the pause between two digits of a DTMF sequence.

return
digit_pause: Range: 0 s to 3 s, Unit: s

get_dtime() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:DTIMe
value: float = driver.source.afRf.generator.dialing.dtmf.get_dtime()
```

Defines the duration of a single digit of a DTMF sequence.

return
digit_time: Range: 0.02 s to 3 s, Unit: s

get_sequence() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SEQuence
value: str = driver.source.afRf.generator.dialing.dtmf.get_sequence()
```

Specifies a digit sequence for the dialing mode DTMF.

return
dtm_fsequence: String with 1 to 42 digits The allowed digits are 0 to 9, A to D, * and # (with user-defined tone table also m) .

get_spause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SPAuse
value: float = driver.source.afRf.generator.dialing.dtmf.get_spause()
```

Defines the duration of a pause between two repetitions of a DTMF sequence.

return
sequence_pause: Range: 0 s to 10 s, Unit: s

get_srepeat() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SREPeat
value: int = driver.source.afRf.generator.dialing.dtmf.get_srepeat()
```

Defines how often a DTMF sequence is repeated.

return
sequence_repeat: Range: 1 to 100

set_dpause(*digit_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:DPAuse
driver.source.afRf.generator.dialing.dtmf.set_dpause(digit_pause = 1.0)
```

Defines the duration of the pause between two digits of a DTMF sequence.

param digit_pause
Range: 0 s to 3 s, Unit: s

set_dtime(*digit_time: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:DTIMe
driver.source.afRf.generator.dialing.dtmf.set_dtime(digit_time = 1.0)
```

Defines the duration of a single digit of a DTMF sequence.

param digit_time
Range: 0.02 s to 3 s, Unit: s

set_sequence(*dtm_fsequence: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SEQuence
driver.source.afRf.generator.dialing.dtmf.set_sequence(dtm_fsequence = '1')
```

Specifies a digit sequence for the dialing mode DTMF.

param dtm_fsequence
String with 1 to 42 digits The allowed digits are 0 to 9, A to D, * and # (with user-defined tone table also m) .

set_spause(*sequence_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SPAuse
driver.source.afRf.generator.dialing.dtmf.set_spause(sequence_pause = 1.0)
```

Defines the duration of a pause between two repetitions of a DTMF sequence.

param sequence_pause
Range: 0 s to 10 s, Unit: s

set_srepeat(*sequence_repeat: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:SREPeat
driver.source.afRf.generator.dialing.dtmf.set_srepeat(sequence_repeat = 1)
```

Defines how often a DTMF sequence is repeated.

param sequence_repeat
Range: 1 to 100

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.dialing.dtmf.clone()
```

Subgroups

6.15.1.1.5.2 Frequency

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:FREQuency:RESet
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:FREQuency
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_value() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:FREQuency
value: List[float] = driver.source.afRf.generator.dialing.dtmf.frequency.get_
↪value()
```

Configures the user-defined tone table for DTMF. To enable the table, see method RsCma.Source.AfRf.Generator.Dialing.Dtmf. UserDefined.enable.

return
dtmf_frequency: Comma-separated list of up to 8 frequencies You can specify fewer than 8 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

reset() → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:FREQuency:RESet
driver.source.afRf.generator.dialing.dtmf.frequency.reset()
```

Triggers a reset of user-defined frequency values to the default frequency values of the DTMF standard.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:FREQuency:RESet
driver.source.afRf.generator.dialing.dtmf.frequency.reset_with_opc()
```

Triggers a reset of user-defined frequency values to the default frequency values of the DTMF standard.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*dtmf_frequency: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:FREQuency
driver.source.afRf.generator.dialing.dtmf.frequency.set_value(dtmf_frequency =␣
↪[1.1, 2.2, 3.3])
```

Configures the user-defined tone table for DTMF. To enable the table, see method RsCma.Source.AfRf.Generator.Dialing.Dtmf. UserDefined.enable.

param dtmf_frequency
Comma-separated list of up to 8 frequencies You can specify fewer than 8 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

6.15.1.1.5.3 UserDefined

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:UDEFined:ENABle
```

class UserDefinedCls

UserDefined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:UDEFined:ENABle
value: bool = driver.source.afRf.generator.dialing.dtmf.userDefined.get_enable()
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Source.AfRf.Generator.Dialing. Dtmf.Frequency.value.

return
dtmf_userdefined: OFF | ON ON: user-defined tone table OFF: default tone table

set_enable(*dtmf_userdefined: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:DTMF:UDEFined:ENABle
driver.source.afRf.generator.dialing.dtmf.userDefined.set_enable(dtmf_
↪userdefined = False)
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Source.AfRf.Generator.Dialing. Dtmf.Frequency.value.

param dtmf_userdefined
OFF | ON ON: user-defined tone table OFF: default tone table

6.15.1.1.5.4 Fdialing

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:TTYPe
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SEQuence
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SREPeat
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SPAuse
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DTIMe
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DPAuse
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DTMode
```

class FdialingCls

Fdialing commands group definition. 11 total commands, 2 Subgroups, 7 group commands

get_dpause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DPAuse
value: float = driver.source.afRf.generator.dialing.fdialing.get_dpause()
```

Defines the duration of the pause between two digits of a free dialing sequence.

return
digit_pause: Range: 0 s to 3 s, Unit: s

get_dt_mode() → DigitTimeMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DTMode
value: enums.DigitTimeMode = driver.source.afRf.generator.dialing.fdialing.get_
↪dt_mode()
```

Defines if all digit times and digit pauses have equal lengths or individual lengths.

return
digit_time_mode: EQUal | INDividual EQUal Equal length of digit times and digit pauses INDividual Individual length of digit times, digit pauses are equal in length.

get_dtime() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DTIMe
value: float = driver.source.afRf.generator.dialing.fdialing.get_dtime()
```

Defines the duration of a single digit of a free-dialing sequence.

return
digit_time: Range: 0.02 s to 3 s, Unit: s

get_sequence() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SEQuence
value: str = driver.source.afRf.generator.dialing.fdialing.get_sequence()
```

Specifies a digit sequence for the mode free dialing.

return
fdialing_sequence: String with 1 to 42 digits The allowed digits are 0 to 9, A to F and m.

get_spause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SPAuse
value: float = driver.source.afRf.generator.dialing.fdialing.get_spause()
```

Defines the duration of a pause between two repetitions of a free-dialing sequence.

return
sequence_pause: Range: 0 s to 10 s, Unit: s

get_srepeat() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SREPeat
value: int = driver.source.afRf.generator.dialing.fdialing.get_srepeat()
```

Defines how often a free-dialing sequence is repeated.

return
sequence_repeat: Range: 1 to 100

get_ttype() → SingDualTonesType

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:TTYPe
value: enums.SingDualTonesType = driver.source.afRf.generator.dialing.fdialing.
↪get_ttype()
```

Selects the tone type for free dialing.

return
tone_type: STONes | DTONes Single tones or dual tones

set_dpause(*digit_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DPAuse
driver.source.afRf.generator.dialing.fdialing.set_dpause(digit_pause = 1.0)
```

Defines the duration of the pause between two digits of a free dialing sequence.

param digit_pause
Range: 0 s to 3 s, Unit: s

set_dt_mode(*digit_time_mode: DigitTimeMode*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DTMode
driver.source.afRf.generator.dialing.fdialing.set_dt_mode(digit_time_mode =␣
↪enums.DigitTimeMode.EQUal)
```

Defines if all digit times and digit pauses have equal lengths or individual lengths.

param digit_time_mode
EQUal | INDividual EQUal Equal length of digit times and digit pauses INDividual Individual length of digit times, digit pauses are equal in length.

set_dtime(*digit_time: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:DTIMe
driver.source.afRf.generator.dialing.fdialing.set_dtime(digit_time = 1.0)
```

Defines the duration of a single digit of a free-dialing sequence.

param digit_time
Range: 0.02 s to 3 s, Unit: s

set_sequence(*fdialing_sequence: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SEQuence
driver.source.afRf.generator.dialing.fdialing.set_sequence(fdialing_sequence =
↪'1')
```

Specifies a digit sequence for the mode free dialing.

param fdialing_sequence
String with 1 to 42 digits The allowed digits are 0 to 9, A to F and m.

set_spause(*sequence_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SPAuse
driver.source.afRf.generator.dialing.fdialing.set_spause(sequence_pause = 1.0)
```

Defines the duration of a pause between two repetitions of a free-dialing sequence.

param sequence_pause
Range: 0 s to 10 s, Unit: s

set_srepeat(*sequence_repeat: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:SREPeat
driver.source.afRf.generator.dialing.fdialing.set_srepeat(sequence_repeat = 1)
```

Defines how often a free-dialing sequence is repeated.

param sequence_repeat
Range: 1 to 100

set_ttype(*tone_type: SingDualTonesType*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:TTYPe
driver.source.afRf.generator.dialing.fdialing.set_ttype(tone_type = enums.
↪SingDualTonesType.DTONes)
```

Selects the tone type for free dialing.

param tone_type
STONes | DTONes Single tones or dual tones

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.dialing.fdialing.clone()
```

Subgroups

6.15.1.1.5.5 Frequency<FrequencyLobe>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.source.afRf.generator.dialing.fdialing.frequency.repcap_frequencyLobe_get()
driver.source.afRf.generator.dialing.fdialing.frequency.repcap_frequencyLobe_set(repcap.
↪FrequencyLobe.Nr1)
```

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:FREQuency:STONe
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: FrequencyLobe, default value after init: FrequencyLobe.Nr1

get_stone() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:FREQuency:STONe
value: List[float] = driver.source.afRf.generator.dialing.fdialing.frequency.
↪get_stone()
```

Assigns frequencies to the digits available for free dialing, tone type single tone.

return
tones_frequency: Comma-separated list of 16 frequencies, assigned to the digits 0, 1, …, 9, A, …, F Specifying fewer frequencies leaves the remaining digits unchanged. Range: 60 Hz to 4000 Hz, Unit: Hz

set_stone(*tones_frequency: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:FREQuency:STONe
driver.source.afRf.generator.dialing.fdialing.frequency.set_stone(tones_
↪frequency = [1.1, 2.2, 3.3])
```

Assigns frequencies to the digits available for free dialing, tone type single tone.

param tones_frequency
Comma-separated list of 16 frequencies, assigned to the digits 0, 1, …, 9, A, …, F Specifying fewer frequencies leaves the remaining digits unchanged. Range: 60 Hz to 4000 Hz, Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.dialing.fdialing.frequency.clone()
```

Subgroups

6.15.1.1.5.6 Dtone

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:FREQuency<Nr>:DTONe
```

class DtoneCls

Dtone commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*frequencyLobe=FrequencyLobe.Default*) → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:FREQuency<Nr>:DTONe
value: List[float] = driver.source.afRf.generator.dialing.fdialing.frequency.
↪dtone.get(frequencyLobe = repcap.FrequencyLobe.Default)
```

Assigns frequencies to the digits available for free dialing, tone type dual tone.

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
tones_frequency: Comma-separated list of 16 frequencies, assigned to the digits 0, 1, ..., 9, A, ..., F Specifying fewer frequencies leaves the remaining digits unchanged. Range: no=1/2: 60 Hz to 1000 Hz / 1200 Hz to 4000 Hz , Unit: Hz

set(*tones_frequency: List[float]*, *frequencyLobe=FrequencyLobe.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:FREQuency<Nr>:DTONe
driver.source.afRf.generator.dialing.fdialing.frequency.dtone.set(tones_
↪frequency = [1.1, 2.2, 3.3], frequencyLobe = repcap.FrequencyLobe.Default)
```

Assigns frequencies to the digits available for free dialing, tone type dual tone.

param tones_frequency
Comma-separated list of 16 frequencies, assigned to the digits 0, 1, ..., 9, A, ..., F Specifying fewer frequencies leaves the remaining digits unchanged. Range: no=1/2: 60 Hz to 1000 Hz / 1200 Hz to 4000 Hz , Unit: Hz

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

6.15.1.1.5.7 Individual

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:INDividual:DTIMe
SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:INDividual:DPAuse
```

class IndividualCls

Individual commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_dpause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:INDividual:DPAuse
value: float = driver.source.afRf.generator.dialing.fdialing.individual.get_
↪dpause()
```

Common length of the pause between two individual digits of a free dialing sequence.

return
digit_pause: Range: 0 to 3, Unit: s

get_dtime() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:INDividual:DTIMe
value: List[float] = driver.source.afRf.generator.dialing.fdialing.individual.
↪get_dtime()
```

Sets individual digit times for digits of the free dialing sequence.

return
digit_time_array: Comma-separated list of individual digit times of the digits of the dialing sequence Unit: s

set_dpause(*digit_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:INDividual:DPAuse
driver.source.afRf.generator.dialing.fdialing.individual.set_dpause(digit_pause␣
↪= 1.0)
```

Common length of the pause between two individual digits of a free dialing sequence.

param digit_pause
Range: 0 to 3, Unit: s

set_dtime(*digit_time_array: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:FDIaling:INDividual:DTIMe
driver.source.afRf.generator.dialing.fdialing.individual.set_dtime(digit_time_
↪array = [1.1, 2.2, 3.3])
```

Sets individual digit times for digits of the free dialing sequence.

param digit_time_array
Comma-separated list of individual digit times of the digits of the dialing sequence Unit: s

6.15.1.1.5.8 Scal

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:STANdard
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SEQuence
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SREPeat
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SPAuse
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:TPAuse
```

class ScalCls

Scal commands group definition. 9 total commands, 3 Subgroups, 5 group commands

get_sequence() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SEQuence
value: str = driver.source.afRf.generator.dialing.scal.get_sequence()
```

Specifies the SELCAL code (without the hyphen) .

return
scal_sequence: String with four letters The allowed letters are A to H, J to M and P to S. A letter must not appear twice. The first two letters and the last two letters must be ordered alphabetically.

get_spause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SPAuse
value: float = driver.source.afRf.generator.dialing.scal.get_spause()
```

Defines the duration of a pause between two repetitions of a SELCAL sequence.

return
sequence_pause: Range: 0.1 s to 10 s, Unit: s

get_srepeat() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SREPeat
value: int = driver.source.afRf.generator.dialing.scal.get_srepeat()
```

Defines how often a SELCAL sequence (two dual tones) is repeated.

return
sequence_repeat: Range: 1 to 100

get_standard() → SelCalStandard

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:STANdard
value: enums.SelCalStandard = driver.source.afRf.generator.dialing.scal.get_
↪standard()
```

Selects the SELCAL standard. Selecting the standard also determines the dual-tone frequencies that the CMA generates when generating a SELCAL dialing signal.

return
standard: SCAL16 | SCAL32 | UDEFined SCAL16 SELCAL16 standard and corre-

sponding frequencies SCAL32 SELCAL32 standard and corresponding frequencies UDEFined User-defined dual-tone frequencies

get_tpause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:TPAuse
value: float = driver.source.afRf.generator.dialing.scal.get_tpause()
```

Defines the duration of the pause between the two dual tones of a SELCAL sequence.

return
tpause: Range: 0.1 s to 3 s, Unit: s

set_sequence(*scal_sequence: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SEQuence
driver.source.afRf.generator.dialing.scal.set_sequence(scal_sequence = '1')
```

Specifies the SELCAL code (without the hyphen) .

param scal_sequence
String with four letters The allowed letters are A to H, J to M and P to S. A letter must not appear twice. The first two letters and the last two letters must be ordered alphabetically.

set_spause(*sequence_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SPAuse
driver.source.afRf.generator.dialing.scal.set_spause(sequence_pause = 1.0)
```

Defines the duration of a pause between two repetitions of a SELCAL sequence.

param sequence_pause
Range: 0.1 s to 10 s, Unit: s

set_srepeat(*sequence_repeat: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:SREPeat
driver.source.afRf.generator.dialing.scal.set_srepeat(sequence_repeat = 1)
```

Defines how often a SELCAL sequence (two dual tones) is repeated.

param sequence_repeat
Range: 1 to 100

set_standard(*standard: SelCalStandard*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:STANdard
driver.source.afRf.generator.dialing.scal.set_standard(standard = enums.
↪SelCalStandard.SCAL16)
```

Selects the SELCAL standard. Selecting the standard also determines the dual-tone frequencies that the CMA generates when generating a SELCAL dialing signal.

param standard
SCAL16 | SCAL32 | UDEFined SCAL16 SELCAL16 standard and corresponding frequencies SCAL32 SELCAL32 standard and corresponding frequencies UDEFined User-defined dual-tone frequencies

set_tpause(*tpause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:TPAuse
driver.source.afRf.generator.dialing.scal.set_tpause(tpause = 1.0)
```

Defines the duration of the pause between the two dual tones of a SELCAL sequence.

param tpause
Range: 0.1 s to 3 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.dialing.scal.clone()
```

Subgroups

6.15.1.1.5.9 Frequency

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:FREQuency:RESet
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:FREQuency
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_value() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:FREQuency
value: List[float] = driver.source.afRf.generator.dialing.scal.frequency.get_
↪value()
```

Configures the user-defined tone table for SELCAL. To enable the table, set as follows: SOURce:AFRF:GEN:DIALing:SCAL:STANdard UDEFined See alsomethod RsCma.Source.AfRf.Generator.Dialing.Scal.standard

return
frequency: Comma-separated list of up to 16 frequencies You can specify fewer than 16 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

reset() → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:FREQuency:RESet
driver.source.afRf.generator.dialing.scal.frequency.reset()
```

Triggers a reset of user-defined frequency values to the default frequency values of the SELCAL standard.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:FREQuency:RESet
driver.source.afRf.generator.dialing.scal.frequency.reset_with_opc()
```

Triggers a reset of user-defined frequency values to the default frequency values of the SELCAL standard.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set_value(*frequency: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:FREQuency
driver.source.afRf.generator.dialing.scal.frequency.set_value(frequency = [1.1,␣
↪2.2, 3.3])
```

Configures the user-defined tone table for SELCAL. To enable the table, set as follows: SOURce:AFRF:GEN:DIALing:SCAL:STANdard UDEFined See alsomethod RsCma.Source.AfRf.Generator.Dialing.Scal.standard

param frequency
Comma-separated list of up to 16 frequencies You can specify fewer than 16 values to configure only the beginning of the tone table. Range: see table , Unit: Hz

6.15.1.1.5.10 Ttime

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:TTIMe:FIRSt
```

class TtimeCls

Ttime commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_first() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:TTIMe:FIRSt
value: float = driver.source.afRf.generator.dialing.scal.ttime.get_first()
```

Defines the duration of a dual tone of a SELCAL sequence.

return
ttime: Range: 0.2 s to 3 s, Unit: s

set_first(*ttime: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:TTIMe:FIRSt
driver.source.afRf.generator.dialing.scal.ttime.set_first(ttime = 1.0)
```

Defines the duration of a dual tone of a SELCAL sequence.

param ttime
Range: 0.2 s to 3 s, Unit: s

6.15.1.1.5.11 UserDefined

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:UDEFined:ENABle
```

class UserDefinedCls

UserDefined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:UDEFined:ENABle
value: bool = driver.source.afRf.generator.dialing.scal.userDefined.get_enable()
```

No command help available

return
user_defined: No help available

set_enable(*user_defined: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SCAL:UDEFined:ENABle
driver.source.afRf.generator.dialing.scal.userDefined.set_enable(user_defined =␣
↪False)
```

No command help available

param user_defined
No help available

6.15.1.1.5.12 SelCall

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:STANdard
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SEQuence
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SREPeat
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SPAuse
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DTIMe
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DPAuse
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DREPeat
```

class SelCallCls

SelCall commands group definition. 10 total commands, 2 Subgroups, 7 group commands

get_dpause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DPAuse
value: float = driver.source.afRf.generator.dialing.selCall.get_dpause()
```

Defines the duration of the pause between two digits of a SelCall sequence.

return
digit_pause: Range: 0 s to 3 s, Unit: s

get_drepeat() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DREPeat
value: bool = driver.source.afRf.generator.dialing.selCall.get_drepeat()
```

Enables or disables the usage of the repeat digit for the tone mode SelCall.

return
digit_repeat: OFF | ON

get_dtime() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DTIMe
value: float = driver.source.afRf.generator.dialing.selCall.get_dtime()
```

Defines the duration of a single digit of a SelCall sequence.

return
digit_time: Range: 0.02 s to 3 s, Unit: s

get_sequence() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SEQuence
value: str = driver.source.afRf.generator.dialing.selCall.get_sequence()
```

Specifies a digit sequence for the dialing mode SelCall.

return
sel_call_sequence: String with five digits - allowed digits are 0 to 9 and A to F If the user-defined tone definition is enabled, you can define 1 to 42 digits and also use the digit m.

get_spause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SPAuse
value: float = driver.source.afRf.generator.dialing.selCall.get_spause()
```

Defines the duration of a pause between two repetitions of a SelCall sequence.

return
sequence_pause: Range: 0 s to 10 s, Unit: s

get_srepeat() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SREPeat
value: int = driver.source.afRf.generator.dialing.selCall.get_srepeat()
```

Defines how often a SelCall sequence is repeated.

return
sequence_repeat: Range: 1 to 100

get_standard() → SelCallStandard

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:STANdard
value: enums.SelCallStandard = driver.source.afRf.generator.dialing.selCall.get_
↪standard()
```

Selects the SelCall standard and thus the tone definition.

return
sel_call_standard: CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei

set_dpause(*digit_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DPAuse
driver.source.afRf.generator.dialing.selCall.set_dpause(digit_pause = 1.0)
```

Defines the duration of the pause between two digits of a SelCall sequence.

param digit_pause
Range: 0 s to 3 s, Unit: s

set_drepeat(*digit_repeat: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DREPeat
driver.source.afRf.generator.dialing.selCall.set_drepeat(digit_repeat = False)
```

Enables or disables the usage of the repeat digit for the tone mode SelCall.

param digit_repeat
OFF | ON

set_dtime(*digit_time: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:DTIMe
driver.source.afRf.generator.dialing.selCall.set_dtime(digit_time = 1.0)
```

Defines the duration of a single digit of a SelCall sequence.

param digit_time
Range: 0.02 s to 3 s, Unit: s

set_sequence(*sel_call_sequence: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SEQuence
driver.source.afRf.generator.dialing.selCall.set_sequence(sel_call_sequence = '1
↪')
```

Specifies a digit sequence for the dialing mode SelCall.

param sel_call_sequence
String with five digits - allowed digits are 0 to 9 and A to F If the user-defined tone definition is enabled, you can define 1 to 42 digits and also use the digit m.

set_spause(*sequence_pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SPAuse
driver.source.afRf.generator.dialing.selCall.set_spause(sequence_pause = 1.0)
```

Defines the duration of a pause between two repetitions of a SelCall sequence.

param sequence_pause
Range: 0 s to 10 s, Unit: s

set_srepeat(*sequence_repeat: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:SREPeat
driver.source.afRf.generator.dialing.selCall.set_srepeat(sequence_repeat = 1)
```

Defines how often a SelCall sequence is repeated.

param sequence_repeat
Range: 1 to 100

set_standard(*sel_call_standard: SelCallStandard*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:STANdard
driver.source.afRf.generator.dialing.selCall.set_standard(sel_call_standard =␣
↪enums.SelCallStandard.CCIR)
```

Selects the SelCall standard and thus the tone definition.

param sel_call_standard
CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.dialing.selCall.clone()
```

Subgroups

6.15.1.1.5.13 Frequency

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:FREQuency:RESet
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:FREQuency
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get(*standard: SelCallStandard*) → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:FREQuency
value: List[float] = driver.source.afRf.generator.dialing.selCall.frequency.
↪get(standard = enums.SelCallStandard.CCIR)
```

Configures the user-defined tone table for a SelCall standard. To enable the table, see method RsCma.Source.AfRf. Generator.Dialing.SelCall.UserDefined.enable.

param standard
CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei Selects the SelCall standard

return
frequency: Comma-separated list of up to 16 frequencies, for digit 0 to F You can specify fewer than 16 values to configure only the beginning of the tone table. The *RST values and ranges depend on the SelCall standard and on the digit. The ranges are approximately: default frequency minus 5% to default frequency plus 5%. Unit: Hz

reset() → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:FREQuency:RESet
driver.source.afRf.generator.dialing.selCall.frequency.reset()
```

Triggers a reset of user-defined frequency values to the default frequency values of the selective-calling standard.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:FREQuency:RESet
driver.source.afRf.generator.dialing.selCall.frequency.reset_with_opc()
```

Triggers a reset of user-defined frequency values to the default frequency values of the selective-calling standard.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

set(*standard: SelCallStandard*, *frequency: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:FREQuency
driver.source.afRf.generator.dialing.selCall.frequency.set(standard = enums.
↪SelCallStandard.CCIR, frequency = [1.1, 2.2, 3.3])
```

Configures the user-defined tone table for a SelCall standard. To enable the table, see method RsCma.Source.AfRf. Generator.Dialing.SelCall.UserDefined.enable.

param standard
CCIR | EEA | EIA | ZVEI1 | ZVEI2 | ZVEI3 | DZVei | PZVei Selects the SelCall standard

param frequency
Comma-separated list of up to 16 frequencies, for digit 0 to F You can specify fewer than 16 values to configure only the beginning of the tone table. The *RST values and ranges depend on the SelCall standard and on the digit. The ranges are approximately: default frequency minus 5% to default frequency plus 5%. Unit: Hz

6.15.1.1.5.14 UserDefined

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:UDEFined:ENABle
```

class UserDefinedCls

UserDefined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:UDEFined:ENABle
value: bool = driver.source.afRf.generator.dialing.selCall.userDefined.get_
↪enable()
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Source.AfRf.Generator.Dialing. SelCall.Frequency.set.

return
user_defined: OFF | ON ON: user-defined tone table OFF: default tone table

set_enable(*user_defined: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIALing:SELCall:UDEFined:ENABle
driver.source.afRf.generator.dialing.selCall.userDefined.set_enable(user_
↪defined = False)
```

Enables or disables the user-defined tone table. The table is configured via method RsCma.Source.AfRf.Generator.Dialing. SelCall.Frequency.set.

param user_defined
OFF | ON ON: user-defined tone table OFF: default tone table

6.15.1.1.6 Digital

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIGital:FILE
```

class DigitalCls

Digital commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_file() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIGital:FILE
value: str = driver.source.afRf.generator.digital.get_file()
```

No command help available

return
arb_file: No help available

set_file(*arb_file: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIGital:FILE
driver.source.afRf.generator.digital.set_file(arb_file = '1')
```

No command help available

param arb_file
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.digital.clone()
```

Subgroups

6.15.1.1.6.1 Rf

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DIGital:RF:ENABle
```

class RfCls

Rf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIGital:RF:ENABle
value: bool = driver.source.afRf.generator.digital.rf.get_enable()
```

No command help available

return
dig_rf_enable: No help available

set_enable(*dig_rf_enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DIGital:RF:ENABle
driver.source.afRf.generator.digital.rf.set_enable(dig_rf_enable = False)
```

No command help available

param dig_rf_enable
No help available

6.15.1.1.7 Dmr

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DMR:PATTern
SOURce:AFRF:GENerator<Instance>:DMR:SVALue
SOURce:AFRF:GENerator<Instance>:DMR:CCODe
SOURce:AFRF:GENerator<Instance>:DMR:SADDress
SOURce:AFRF:GENerator<Instance>:DMR:GADDress
SOURce:AFRF:GENerator<Instance>:DMR:MODE
SOURce:AFRF:GENerator<Instance>:DMR:SDEViation
SOURce:AFRF:GENerator<Instance>:DMR:SRATe
SOURce:AFRF:GENerator<Instance>:DMR:FILTer
SOURce:AFRF:GENerator<Instance>:DMR:ROFactor
```

class DmrCls

Dmr commands group definition. 10 total commands, 0 Subgroups, 10 group commands

get_ccode() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:CCODe
value: int = driver.source.afRf.generator.dmr.get_ccode()
```

Defines the color code to be signaled to the DUT, for DMR.

return
ccode: Range: 0 to 15

get_filter_py() → PulseShapingFilter

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:FILTer
value: enums.PulseShapingFilter = driver.source.afRf.generator.dmr.get_filter_
↪py()
```

Queries the filter type used for pulse shaping for DMR.

return
filter_py: RRC

get_gaddress() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:GADDress
value: float = driver.source.afRf.generator.dmr.get_gaddress()
```

Configures the group address to be signaled to the DUT, for DMR.

return
gaddress: Range: 0 to 16777215

get_mode() → FskMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:MODE
value: enums.FskMode = driver.source.afRf.generator.dmr.get_mode()
```

Queries the modulation type used for DMR.

return
mode: FSK4

get_pattern() → DmrPattern

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:PATTern
value: enums.DmrPattern = driver.source.afRf.generator.dmr.get_pattern()
```

Selects the bit pattern to be transmitted as payload for DMR.

return
pattern: P1031 | SILence | PRBS9 | O153 | C153 P1031 Audio tone with a frequency of 1031 Hz. SILence The payload contains silence. PRBS9 Pseudo-random binary sequence with 511 bits (29-1) . O153 Modified 'PRBS9' pattern optimized for bit error rate (BER) measurements. C153 Modified 'PRBS9' pattern with bit errors after every 100 bits resulting in BER = 1 %. Use this pattern to verify that your DUT can calculate bit errors.

get_ro_factor() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:ROFactor
value: float = driver.source.afRf.generator.dmr.get_ro_factor()
```

Queries the roll-off factor of the filter used for pulse shaping for DMR.

return
ro_factor: Range: 0.2 to 0.2

get_saddress() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:SADDress
value: float = driver.source.afRf.generator.dmr.get_saddress()
```

Configures the source address to be signaled to the DUT, for DMR.

return
saddress: Range: 0 to 16777215

get_standard_dev() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:SDEViation
value: List[float] = driver.source.afRf.generator.dmr.get_standard_dev()
```

Queries the frequency deviations used for 4FSK modulation, for DMR.

return
sdeviation: List of four frequency deviations, for the symbols 01, 00, 10, 11. Range: -2000 Hz to 2000 Hz, Unit: Hz

get_svalue() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:SVALue
value: str = driver.source.afRf.generator.dmr.get_svalue()
```

Specifies the 9-bit seed value for the PRBS generator, for DMR.

return
svalue: Range: #H0 to #H1FF

get_symbol_rate() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:SRATe
value: float = driver.source.afRf.generator.dmr.get_symbol_rate()
```

Queries the symbol rate for DMR.

return
srate: Range: 4800 symbol/s to 4800 symbol/s , Unit: symbol/s

set_ccode(*ccode: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:CCODe
driver.source.afRf.generator.dmr.set_ccode(ccode = 1)
```

Defines the color code to be signaled to the DUT, for DMR.

param ccode
Range: 0 to 15

set_gaddress(*gaddress: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:GADDress
driver.source.afRf.generator.dmr.set_gaddress(gaddress = 1.0)
```

Configures the group address to be signaled to the DUT, for DMR.

param gaddress
Range: 0 to 16777215

set_pattern(*pattern: DmrPattern*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:PATTern
driver.source.afRf.generator.dmr.set_pattern(pattern = enums.DmrPattern.BPRB15)
```

Selects the bit pattern to be transmitted as payload for DMR.

param pattern
P1031 | SILence | PRBS9 | O153 | C153 P1031 Audio tone with a frequency of 1031 Hz. SILence The payload contains silence. PRBS9 Pseudo-random binary sequence with 511 bits (29-1) . O153 Modified 'PRBS9' pattern optimized for bit error rate (BER) measurements. C153 Modified 'PRBS9' pattern with bit errors after every 100 bits resulting in BER = 1 %. Use this pattern to verify that your DUT can calculate bit errors.

set_saddress(*saddress: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:SADDress
driver.source.afRf.generator.dmr.set_saddress(saddress = 1.0)
```

Configures the source address to be signaled to the DUT, for DMR.

param saddress
Range: 0 to 16777215

set_svalue(*svalue: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DMR:SVALue
driver.source.afRf.generator.dmr.set_svalue(svalue = r1)
```

Specifies the 9-bit seed value for the PRBS generator, for DMR.

param svalue
Range: #H0 to #H1FF

6.15.1.1.8 Dpmr

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DPMR:PATTern
SOURce:AFRF:GENerator<Instance>:DPMR:SVALue
SOURce:AFRF:GENerator<Instance>:DPMR:SID
SOURce:AFRF:GENerator<Instance>:DPMR:DID
SOURce:AFRF:GENerator<Instance>:DPMR:PTPeer
SOURce:AFRF:GENerator<Instance>:DPMR:EMERgency
```

(continues on next page)

(continued from previous page)

```
SOURce:AFRF:GENerator<Instance>:DPMR:MODE
SOURce:AFRF:GENerator<Instance>:DPMR:SDEViation
SOURce:AFRF:GENerator<Instance>:DPMR:SRATe
SOURce:AFRF:GENerator<Instance>:DPMR:FILTer
SOURce:AFRF:GENerator<Instance>:DPMR:ROFactor
```

class DpmrCls

Dpmr commands group definition. 13 total commands, 1 Subgroups, 11 group commands

get_did() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:DID
value: str = driver.source.afRf.generator.dpmr.get_did()
```

Configures the destination ID, for DPMR.

return
did: No help available

get_emergency() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:EMERgency
value: bool = driver.source.afRf.generator.dpmr.get_emergency()
```

Configures the emergency bit to be signaled to the DUT, for DPMR.

return
emergency: OFF | ON

get_filter_py() → PulseShapingFilter

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:FILTer
value: enums.PulseShapingFilter = driver.source.afRf.generator.dpmr.get_filter_
↪py()
```

Queries the filter type used for pulse shaping for the DPMR standard.

return
filter_py: RRC Root-raised-cosine filter

get_mode() → FskMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:MODE
value: enums.FskMode = driver.source.afRf.generator.dpmr.get_mode()
```

Queries the modulation type used for DPMR.

return
mode: FSK2 | FSK4

get_pattern() → NxdnPatternB

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:PATTern
value: enums.NxdnPatternB = driver.source.afRf.generator.dpmr.get_pattern()
```

Selects the bit pattern to be transmitted as payload for DPMR.

return
pattern: RSYR | RLD | R1031 | RA1 | RA0 | R10A | RBRB9 | RBRB15 | CUST

get_pt_peer() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:PTPeer
value: bool = driver.source.afRf.generator.dpmr.get_pt_peer()
```

Configures the 'Peer to Peer' bit for the communication between DUTs (DPMR mode 1) .

return
emergency: OFF | ON

get_ro_factor() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:ROFactor
value: float = driver.source.afRf.generator.dpmr.get_ro_factor()
```

Queries the roll-off factor of the filter used for pulse shaping, for DPMR.

return
ro_factor: Range: 0 to 1

get_sid() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:SID
value: str = driver.source.afRf.generator.dpmr.get_sid()
```

Configures the source ID, for DPMR.

return
sid: No help available

get_standard_dev() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:SDEViation
value: List[float] = driver.source.afRf.generator.dpmr.get_standard_dev()
```

Queries the frequency deviations used for 4FSK modulation, for DPMR.

return
sdeviation: List of four frequency deviations, for the symbols 01, 00, 10, 11. Range: -2000 Hz to 2000 Hz, Unit: Hz

get_svalue() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:SVALue
value: str = driver.source.afRf.generator.dpmr.get_svalue()
```

Specifies the 9-bit seed value for the PRBS generator, for DPMR.

return
svalue: Range: #H0 to #H1FF

get_symbol_rate() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:SRATe
value: float = driver.source.afRf.generator.dpmr.get_symbol_rate()
```

Queries the symbol rate for DPMR.

return
srate: Range: 0 symbol/s to 100E+6 symbol/s, Unit: bit/s

set_did(*did: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:DID
driver.source.afRf.generator.dpmr.set_did(did = '1')
```

Configures the destination ID, for DPMR.

param did
No help available

set_emergency(*emergency: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:EMERgency
driver.source.afRf.generator.dpmr.set_emergency(emergency = False)
```

Configures the emergency bit to be signaled to the DUT, for DPMR.

param emergency
OFF | ON

set_pattern(*pattern: NxdnPatternB*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:PATTern
driver.source.afRf.generator.dpmr.set_pattern(pattern = enums.NxdnPatternB.CUST)
```

Selects the bit pattern to be transmitted as payload for DPMR.

param pattern
RSYR | RLD | R1031 | RA1 | RA0 | R10A | RBRB9 | RBRB15 | CUST

set_pt_peer(*emergency: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:PTPeer
driver.source.afRf.generator.dpmr.set_pt_peer(emergency = False)
```

Configures the 'Peer to Peer' bit for the communication between DUTs (DPMR mode 1) .

param emergency
OFF | ON

set_sid(*sid: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:SID
driver.source.afRf.generator.dpmr.set_sid(sid = '1')
```

Configures the source ID, for DPMR.

param sid
No help available

set_svalue(*svalue: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:SVALue
driver.source.afRf.generator.dpmr.set_svalue(svalue = r1)
```

Specifies the 9-bit seed value for the PRBS generator, for DPMR.

param svalue
Range: #H0 to #H1FF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.dpmr.clone()
```

Subgroups

6.15.1.1.8.1 Ccode

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:DPMR:CCODe:CALCulation
SOURce:AFRF:GENerator<Instance>:DPMR:CCODe
```

class CcodeCls

Ccode commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_calculation() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:CCODe:CALCulation
value: bool = driver.source.afRf.generator.dpmr.ccode.get_calculation()
```

Enables or disables the calculation of the channel code, for DPMR.

return
calculation: OFF | ON

get_value() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:CCODe
value: int = driver.source.afRf.generator.dpmr.ccode.get_value()
```

Defines the channel code to be signaled to the DUT, for DPMR.

return
ccode: Range: 0 to 63

set_calculation(*calculation: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:CCODe:CALCulation
driver.source.afRf.generator.dpmr.ccode.set_calculation(calculation = False)
```

Enables or disables the calculation of the channel code, for DPMR.

param calculation
OFF | ON

set_value(*ccode: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:DPMR:CCODe
driver.source.afRf.generator.dpmr.ccode.set_value(ccode = 1)
```

Defines the channel code to be signaled to the DUT, for DPMR.

param ccode
Range: 0 to 63

6.15.1.1.9 FilterPy

class FilterPyCls

FilterPy commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.filterPy.clone()
```

Subgroups

6.15.1.1.9.1 Rf

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:FILTer:RF:PEMPhasis
SOURce:AFRF:GENerator<Instance>:FILTer:RF:HPASs
SOURce:AFRF:GENerator<Instance>:FILTer:RF:LPASs
```

class RfCls

Rf commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_hpass() → HighpassFilter

```
# SCPI: SOURce:AFRF:GENerator<Instance>:FILTer:RF:HPASs
value: enums.HighpassFilter = driver.source.afRf.generator.filterPy.rf.get_
↪hpass()
```

Configures the highpass filter.

return
highpass_filter: OFF | F300 OFF Filter disabled F300 Cutoff frequency 300 Hz

get_lpass() → LowpassFilter

```
# SCPI: SOURce:AFRF:GENerator<Instance>:FILTer:RF:LPASs
value: enums.LowpassFilter = driver.source.afRf.generator.filterPy.rf.get_
↪lpass()
```

Configures the lowpass filter.

return
lowpass_filter: OFF | F3K | F4K | F15K OFF Filter disabled F3K, F4K, F15K Cutoff frequency 3 kHz / 4 kHz / 15 kHz

get_pemphasis() → PreDeEmphasis

```
# SCPI: SOURce:AFRF:GENerator<Instance>:FILTer:RF:PEMPhasis
value: enums.PreDeEmphasis = driver.source.afRf.generator.filterPy.rf.get_
↪pemphasis()
```

Configures the pre-emphasis filter.

return
preemphasis: OFF | T50 | T75 | T750 OFF Filter disabled T50, T75, T750 Time constant 50 us / 75 us / 750 us

set_hpass(*highpass_filter: HighpassFilter*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:FILTer:RF:HPASs
driver.source.afRf.generator.filterPy.rf.set_hpass(highpass_filter = enums.
↪HighpassFilter.F300)
```

Configures the highpass filter.

param highpass_filter
OFF | F300 OFF Filter disabled F300 Cutoff frequency 300 Hz

set_lpass(*lowpass_filter: LowpassFilter*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:FILTer:RF:LPASs
driver.source.afRf.generator.filterPy.rf.set_lpass(lowpass_filter = enums.
↪LowpassFilter.F15K)
```

Configures the lowpass filter.

param lowpass_filter
OFF | F3K | F4K | F15K OFF Filter disabled F3K, F4K, F15K Cutoff frequency 3 kHz / 4 kHz / 15 kHz

set_pemphasis(*preemphasis: PreDeEmphasis*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:FILTer:RF:PEMPhasis
driver.source.afRf.generator.filterPy.rf.set_pemphasis(preemphasis = enums.
↪PreDeEmphasis.OFF)
```

Configures the pre-emphasis filter.

param preemphasis
OFF | T50 | T75 | T750 OFF Filter disabled T50, T75, T750 Time constant 50 us / 75 us / 750 us

6.15.1.1.10 Interferer

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IFERer:MODE
SOURce:AFRF:GENerator<Instance>:IFERer:DFRequency
SOURce:AFRF:GENerator<Instance>:IFERer:DLEVel
```

class InterfererCls

Interferer commands group definition. 9 total commands, 3 Subgroups, 3 group commands

get_dfrequency() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:DFRequency
value: float = driver.source.afRf.generator.interferer.get_dfrequency()
```

Specifies the center RF carrier frequency of the interferer. The frequency is specified as offset value relative to the center carrier frequency of the wanted signal, configured via method RsCma.Source.AfRf.Generator.RfSettings.frequency.

return
frequency: Range: -10 MHz to 10 MHz, Unit: Hz

get_dlevel() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:DLEVel
value: float = driver.source.afRf.generator.interferer.get_dlevel()
```

Sets the RMS level of the interferer RF signal. The level is specified as offset value relative to the level of the wanted signal, configured via method RsCma.Source.AfRf.Generator.RfSettings.level.

return
level: Range: -80 dB to 80 dB, Unit: dB

get_mode() → InterfererMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODE
value: enums.InterfererMode = driver.source.afRf.generator.interferer.get_mode()
```

Selects the interferer signal mode. The interferer signal can be an unmodulated CW signal or a modulated signal, carrying a single tone.

return
interferer_mode: NONE | CW | FM | PM | AM NONE Interferer signal disabled CW Unmodulated RF carrier signal FM, PM, AM Frequency / phase / amplitude modulation

set_dfrequency(*frequency: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:DFRequency
driver.source.afRf.generator.interferer.set_dfrequency(frequency = 1.0)
```

Specifies the center RF carrier frequency of the interferer. The frequency is specified as offset value relative to the center carrier frequency of the wanted signal, configured via method RsCma.Source.AfRf.Generator.RfSettings.frequency.

param frequency
Range: -10 MHz to 10 MHz, Unit: Hz

set_dlevel(*level: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:DLEVel
driver.source.afRf.generator.interferer.set_dlevel(level = 1.0)
```

Sets the RMS level of the interferer RF signal. The level is specified as offset value relative to the level of the wanted signal, configured via method RsCma.Source.AfRf.Generator.RfSettings.level.

param level
Range: -80 dB to 80 dB, Unit: dB

set_mode(*interferer_mode: InterfererMode*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODE
driver.source.afRf.generator.interferer.set_mode(interferer_mode = enums.
↪InterfererMode.AM)
```

Selects the interferer signal mode. The interferer signal can be an unmodulated CW signal or a modulated signal, carrying a single tone.

param interferer_mode
NONE | CW | FM | PM | AM NONE Interferer signal disabled CW Unmodulated RF carrier signal FM, PM, AM Frequency / phase / amplitude modulation

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.interferer.clone()
```

Subgroups

6.15.1.1.10.1 Af

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IFERer:AF:ENABle
SOURce:AFRF:GENerator<Instance>:IFERer:AF:FREQuency
```

class AfCls
Af commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:AF:ENABle
value: bool = driver.source.afRf.generator.interferer.af.get_enable()
```

Enables or disables a single tone. If the tone is disabled, the interferer is unmodulated, even if a modulation mode has been configured.

return
enable: OFF | ON

get_frequency() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:AF:FREQuency
value: float = driver.source.afRf.generator.interferer.af.get_frequency()
```

Configures the frequency of a single tone, that can be added to the interferer.

return
frequency: Range: 0 Hz to 21 kHz, Unit: Hz

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:AF:ENABle
driver.source.afRf.generator.interferer.af.set_enable(enable = False)
```

Enables or disables a single tone. If the tone is disabled, the interferer is unmodulated, even if a modulation mode has been configured.

param enable
OFF | ON

set_frequency(*frequency: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:AF:FREQuency
driver.source.afRf.generator.interferer.af.set_frequency(frequency = 1.0)
```

Configures the frequency of a single tone, that can be added to the interferer.

param frequency
Range: 0 Hz to 21 kHz, Unit: Hz

6.15.1.1.10.2 Modulator

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:FDEViation
SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:PDEViation
SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:MDEPth
```

class ModulatorCls

Modulator commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_fdeviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:FDEViation
value: float = driver.source.afRf.generator.interferer.modulator.get_
↪fdeviation()
```

Specifies the maximum frequency deviation for the FM interferer mode.

return
freq_deviation: Range: 0 Hz to 100 kHz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:MDEPth
value: float = driver.source.afRf.generator.interferer.modulator.get_mod_depth()
```

Specifies the modulation depth for the AM interferer mode.

return
modulation_depth: Range: 0 % to 100 %, Unit: %

get_pdeviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:PDEViation
value: float = driver.source.afRf.generator.interferer.modulator.get_
↪pdeviation()
```

Specifies the maximum phase deviation for the PM interferer mode.

return
phase_deviation: Range: 0 rad to 10 rad, Unit: rad

set_fdeviation(*freq_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:FDEViation
driver.source.afRf.generator.interferer.modulator.set_fdeviation(freq_deviation␣
↪= 1.0)
```

Specifies the maximum frequency deviation for the FM interferer mode.

param freq_deviation
Range: 0 Hz to 100 kHz, Unit: Hz

set_mod_depth(*modulation_depth: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:MDEPth
driver.source.afRf.generator.interferer.modulator.set_mod_depth(modulation_
↪depth = 1.0)
```

Specifies the modulation depth for the AM interferer mode.

param modulation_depth
Range: 0 % to 100 %, Unit: %

set_pdeviation(*phase_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:MODulator:PDEViation
driver.source.afRf.generator.interferer.modulator.set_pdeviation(phase_
↪deviation = 1.0)
```

Specifies the maximum phase deviation for the PM interferer mode.

param phase_deviation
Range: 0 rad to 10 rad, Unit: rad

6.15.1.1.10.3 Rf

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IFERer:RF:ENABle
```

class RfCls

Rf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:RF:ENABle
value: bool = driver.source.afRf.generator.interferer.rf.get_enable()
```

Enables or disables the interferer signal.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IFERer:RF:ENABle
driver.source.afRf.generator.interferer.rf.set_enable(enable = False)
```

Enables or disables the interferer signal.

param enable
OFF | ON

6.15.1.1.11 InternalGenerator<InternalGen>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.source.afRf.generator.internalGenerator.repcap_internalGen_get()
driver.source.afRf.generator.internalGenerator.repcap_internalGen_set(repcap.InternalGen.
↪Nr1)
```

class InternalGeneratorCls

InternalGenerator commands group definition. 20 total commands, 10 Subgroups, 0 group commands Repeated Capability: InternalGen, default value after init: InternalGen.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.clone()
```

Subgroups

6.15.1.1.11.1 Dialing

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing:STARt
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing
```

class DialingCls

Dialing commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get(*internalGen=InternalGen.Default*) → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing
value: bool = driver.source.afRf.generator.internalGenerator.dialing.
↪get(internalGen = repcap.InternalGen.Default)
```

Starts or stops dialing a digit sequence. This command is relevant for dialing modes like SELCAL, DTMF, SelCall or free dialing. For dialing measurements, ensure a delay between starting the measurement and dialing the sequence via this command. Otherwise, the measurement misses the first tones and fails.

INTRO_CMD_HELP: Required delays depending on the input path:

- AF and SPDIF paths: 400 ms
- RF path: FM/PM - 850 ms AM/USB/LSB - 1.0 ms to 1.2 ms

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
dialing_state: OFF | ON

set(*dialing_state: bool*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing
driver.source.afRf.generator.internalGenerator.dialing.set(dialing_state =␣
↪False, internalGen = repcap.InternalGen.Default)
```

Starts or stops dialing a digit sequence. This command is relevant for dialing modes like SELCAL, DTMF, SelCall or free dialing. For dialing measurements, ensure a delay between starting the measurement and dialing the sequence via this command. Otherwise, the measurement misses the first tones and fails.

INTRO_CMD_HELP: Required delays depending on the input path:

- AF and SPDIF paths: 400 ms
- RF path: FM/PM - 850 ms AM/USB/LSB - 1.0 ms to 1.2 ms

param dialing_state
OFF | ON

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

start(*internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing:STARt
driver.source.afRf.generator.internalGenerator.dialing.start(internalGen =␣
↪repcap.InternalGen.Default)
```

No command help available

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

start_with_opc(*internalGen=InternalGen.Default*, *opc_timeout_ms: int = -1*) → None

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.dialing.clone()
```

Subgroups

6.15.1.1.11.2 Mode

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → DialingMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing:MODE
value: enums.DialingMode = driver.source.afRf.generator.internalGenerator.
↪dialing.mode.get(internalGen = repcap.InternalGen.Default)
```

Selects the dialing mode of an internal audio generator. This command is only relevant for non-dialing tone modes, for example tone mode 'single tone' plus dialing mode 'DTMF'.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
tone_type: DTMF | SELCall | FDIaling | SCAL DTMF DTMF sequence SELCall SelCall selective calling FDIaling Free dialing SCAL SELCAL selective calling

set(*tone_type: DialingMode*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DIALing:MODE
driver.source.afRf.generator.internalGenerator.dialing.mode.set(tone_type =␣
↪enums.DialingMode.DTMF, internalGen = repcap.InternalGen.Default)
```

Selects the dialing mode of an internal audio generator. This command is only relevant for non-dialing tone modes, for example tone mode 'single tone' plus dialing mode 'DTMF'.

param tone_type
DTMF | SELCall | FDIaling | SCAL DTMF DTMF sequence SELCall SelCall selective calling FDIaling Free dialing SCAL SELCAL selective calling

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.3 Dtone

class DtoneCls

Dtone commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.dtone.clone()
```

Subgroups

6.15.1.1.11.4 Frequency

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DTONe:FREQuency
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FrequencyStruct

Response structure. Fields:

- Frequency_1: float: Range: 0 Hz to 21 kHz, Unit: Hz
- Frequency_2: float: Range: 0 Hz to 21 kHz, Unit: Hz

get(*internalGen=InternalGen.Default*) → FrequencyStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DTONe:FREQuency
value: FrequencyStruct = driver.source.afRf.generator.internalGenerator.dtone.
↪frequency.get(internalGen = repcap.InternalGen.Default)
```

Configures the frequencies of a dual-tone signal, generated by an internal audio generator.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
structure: for return value, see the help for FrequencyStruct structure arguments.

set(*frequency_1: float*, *frequency_2: float*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:DTONe:FREQuency
driver.source.afRf.generator.internalGenerator.dtone.frequency.set(frequency_1␣
↪= 1.0, frequency_2 = 1.0, internalGen = repcap.InternalGen.Default)
```

Configures the frequencies of a dual-tone signal, generated by an internal audio generator.

param frequency_1
Range: 0 Hz to 21 kHz, Unit: Hz

param frequency_2
Range: 0 Hz to 21 kHz, Unit: Hz

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.5 Enable

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:ENABle
value: bool = driver.source.afRf.generator.internalGenerator.enable.
↪get(internalGen = repcap.InternalGen.Default)
```

Queries whether an internal audio generator is assigned to an audio output path.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
enable: OFF | ON OFF Generator disabled / not assigned to an output path ON Generator enabled / assigned to an output path

6.15.1.1.11.6 First

class FirstCls

First commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.first.clone()
```

Subgroups

6.15.1.1.11.7 MultiTone

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator:FIRSt:MTONe:TLEVel
```

class MultiToneCls

MultiTone commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tlevel() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:FIRSt:MTONe:TLEVel
value: List[float] = driver.source.afRf.generator.internalGenerator.first.
↪multiTone.get_tlevel()
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

return
tlevel: Range: 0 % to 100 %, Unit: %

set_tlevel(*tlevel: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:FIRSt:MTONe:TLEVel
driver.source.afRf.generator.internalGenerator.first.multiTone.set_
↪tlevel(tlevel = [1.1, 2.2, 3.3])
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

param tlevel
Range: 0 % to 100 %, Unit: %

6.15.1.1.11.8 Fourth

class FourthCls

Fourth commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.fourth.clone()
```

Subgroups

6.15.1.1.11.9 MultiTone

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator:FOURth:MTONe:TLEVel
```

class MultiToneCls

MultiTone commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tlevel() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:FOURth:MTONe:TLEVel
value: List[float] = driver.source.afRf.generator.internalGenerator.fourth.
↪multiTone.get_tlevel()
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

return
tlevel: Range: 0 % to 100 %, Unit: %

set_tlevel(*tlevel: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:FOURth:MTONe:TLEVel
driver.source.afRf.generator.internalGenerator.fourth.multiTone.set_
↪tlevel(tlevel = [1.1, 2.2, 3.3])
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

param tlevel
Range: 0 % to 100 %, Unit: %

6.15.1.1.11.10 Frequency

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:FREQuency
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:FREQuency
value: float = driver.source.afRf.generator.internalGenerator.frequency.
↪get(internalGen = repcap.InternalGen.Default)
```

Configures the frequency of a single tone or square signal, generated by an internal audio generator. The maximum frequency for square signals is 4000 Hz.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
frequency: Range: 0 Hz to 21 kHz, Unit: Hz

set(*frequency: float*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:FREQuency
driver.source.afRf.generator.internalGenerator.frequency.set(frequency = 1.0,␣
↪internalGen = repcap.InternalGen.Default)
```

Configures the frequency of a single tone or square signal, generated by an internal audio generator. The maximum frequency for square signals is 4000 Hz.

param frequency
Range: 0 Hz to 21 kHz, Unit: Hz

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.11 MultiTone

class MultiToneCls

MultiTone commands group definition. 9 total commands, 7 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.multiTone.clone()
```

Subgroups

6.15.1.1.11.12 Crest

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:CRESt
```

class CrestCls

Crest commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → CrestFactor

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:CRESt
value: enums.CrestFactor = driver.source.afRf.generator.internalGenerator.
↪multiTone.crest.get(internalGen = repcap.InternalGen.Default)
```

Configures the crest factor for multitone signal generation.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
crest_factor: MAXimum | LOW

set(*crest_factor: CrestFactor*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:CRESt
driver.source.afRf.generator.internalGenerator.multiTone.crest.set(crest_factor␣
↪= enums.CrestFactor.LOW, internalGen = repcap.InternalGen.Default)
```

Configures the crest factor for multitone signal generation.

param crest_factor
MAXimum | LOW

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.13 Enable

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → List[bool]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:ENABle
value: List[bool] = driver.source.afRf.generator.internalGenerator.multiTone.
↪enable.get(internalGen = repcap.InternalGen.Default)
```

Enables or disables the tone list entries for multitone generation.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
tone_status: OFF | ON Comma-separated list of up to 20 values, tone 1 to tone 20 You can specify fewer than 20 values to configure only the beginning of the tone list.

set(*tone_status: List[bool]*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:ENABle
driver.source.afRf.generator.internalGenerator.multiTone.enable.set(tone_status␣
↪= [True, False, True], internalGen = repcap.InternalGen.Default)
```

Enables or disables the tone list entries for multitone generation.

param tone_status
OFF | ON Comma-separated list of up to 20 values, tone 1 to tone 20 You can specify fewer than 20 values to configure only the beginning of the tone list.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.14 Frequency

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:FREQuency
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:FREQuency
value: List[float] = driver.source.afRf.generator.internalGenerator.multiTone.
↪frequency.get(internalGen = repcap.InternalGen.Default)
```

Configures the frequencies of a multitone signal.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
frequency: Comma-separated list of up to 20 frequencies, tone 1 to tone 20 You can specify fewer than 20 values to configure only the beginning of the tone list. Range: 20 Hz to 21 kHz, Unit: Hz

set(*frequency: List[float]*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:FREQuency
driver.source.afRf.generator.internalGenerator.multiTone.frequency.
↪set(frequency = [1.1, 2.2, 3.3], internalGen = repcap.InternalGen.Default)
```

Configures the frequencies of a multitone signal.

param frequency
Comma-separated list of up to 20 frequencies, tone 1 to tone 20 You can specify fewer than 20 values to configure only the beginning of the tone list. Range: 20 Hz to 21 kHz, Unit: Hz

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.multiTone.frequency.clone()
```

Subgroups

6.15.1.1.11.15 Auto

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:FREQuency:AUTO
```

class AutoCls

Auto commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class AutoStruct

Response structure. Fields:

- Start_Freq: float: Frequency for tone 1 in the multitone list Range: 20 Hz to 20 kHz, Unit: Hz
- Freq_Increment: float: Frequency increment for subsequent tones in the list Range: 1 Hz to 20 kHz, Unit: Hz

get(*internalGen=InternalGen.Default*) → AutoStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:FREQuency:AUTO
value: AutoStruct = driver.source.afRf.generator.internalGenerator.multiTone.
↪frequency.auto.get(internalGen = repcap.InternalGen.Default)
```

Configures increasing equidistant frequencies for multitone generation.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
structure: for return value, see the help for AutoStruct structure arguments.

set(*start_freq: float*, *freq_increment: float*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:FREQuency:AUTO
driver.source.afRf.generator.internalGenerator.multiTone.frequency.auto.
↪set(start_freq = 1.0, freq_increment = 1.0, internalGen = repcap.InternalGen.
↪Default)
```

Configures increasing equidistant frequencies for multitone generation.

param start_freq
Frequency for tone 1 in the multitone list Range: 20 Hz to 20 kHz, Unit: Hz

param freq_increment
Frequency increment for subsequent tones in the list Range: 1 Hz to 20 kHz, Unit: Hz

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.16 Ilevel

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:ILEVel
```

class IlevelCls

Ilevel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:ILEVel
value: List[float] = driver.source.afRf.generator.internalGenerator.multiTone.
↪ilevel.get(internalGen = repcap.InternalGen.Default)
```

Configures the levels of all tones of a multitone signal for edit mode INDividual. In edit mode TOTal, you can only query the levels, but not configure them.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
ilevel: Comma-separated list of up to 20 levels, for tone 1 to tone 20 You can specify fewer than 20 values to configure only the beginning of the tone list.

set(*ilevel: List[float]*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:ILEVel
driver.source.afRf.generator.internalGenerator.multiTone.ilevel.set(ilevel = [1.
↪1, 2.2, 3.3], internalGen = repcap.InternalGen.Default)
```

Configures the levels of all tones of a multitone signal for edit mode INDividual. In edit mode TOTal, you can only query the levels, but not configure them.

param ilevel
Comma-separated list of up to 20 levels, for tone 1 to tone 20 You can specify fewer than 20 values to configure only the beginning of the tone list.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.17 Level

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → LevelEditMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:LEVel
value: enums.LevelEditMode = driver.source.afRf.generator.internalGenerator.
↪multiTone.level.get(internalGen = repcap.InternalGen.Default)
```

Selects an edit mode for multitone level configuration.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
level_edit_mode: TOTal | INDividual TOTal All tones have the same level. To configure the total level, see: method RsCma.Source.AfRf.Generator.InternalGenerator.MultiTone.Tlevel.set INDividual The level of each tone is configured separately, see: method RsCma.Source.AfRf.Generator.InternalGenerator.MultiTone.Ilevel.set

set(*level_edit_mode: LevelEditMode*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:LEVel
driver.source.afRf.generator.internalGenerator.multiTone.level.set(level_edit_
↪mode = enums.LevelEditMode.INDividual, internalGen = repcap.InternalGen.
↪Default)
```

Selects an edit mode for multitone level configuration.

param level_edit_mode
TOTal | INDividual TOTal All tones have the same level. To configure the total level, see: method RsCma.Source.AfRf.Generator.InternalGenerator.MultiTone.Tlevel.set INDividual The level of each tone is configured separately, see: method RsCma.Source.AfRf.Generator.InternalGenerator.MultiTone.Ilevel.set

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.18 Tlevel

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TLEVel
```

class TlevelCls

Tlevel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TLEVel
value: List[float] = driver.source.afRf.generator.internalGenerator.multiTone.
↪tlevel.get(internalGen = repcap.InternalGen.Default)
```

Sets the total level of a multitone signal for edit mode TOTal. In edit mode INDividual, you can only query the total level, but not configure it.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
tlevel: Total level

set(*tlevel: List[float]*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TLEVel
driver.source.afRf.generator.internalGenerator.multiTone.tlevel.set(tlevel = [1.
↪1, 2.2, 3.3], internalGen = repcap.InternalGen.Default)
```

Sets the total level of a multitone signal for edit mode TOTal. In edit mode INDividual, you can only query the total level, but not configure it.

param tlevel
Total level

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.19 Tone<ToneNumber>

RepCap Settings

```
# Range: Nr1 .. Nr20
rc = driver.source.afRf.generator.internalGenerator.multiTone.tone.repcap_toneNumber_
↪get()
driver.source.afRf.generator.internalGenerator.multiTone.tone.repcap_toneNumber_
↪set(repcap.ToneNumber.Nr1)
```

class ToneCls

Tone commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: ToneNumber, default value after init: ToneNumber.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.multiTone.tone.clone()
```

Subgroups

6.15.1.1.11.20 All

class AllCls

All commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.multiTone.tone.all.clone()
```

Subgroups

6.15.1.1.11.21 Enable

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TONE:ALL:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*tone_status: bool*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TONE:ALL:ENABle
driver.source.afRf.generator.internalGenerator.multiTone.tone.all.enable.
↪set(tone_status = False, internalGen = repcap.InternalGen.Default)
```

Enables or disables all tone list entries for multitone generation.

param tone_status
OFF | ON

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.11.22 Enable

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TONE<no>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*, *toneNumber=ToneNumber.Default*) → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TONE<no>:ENABle
value: bool = driver.source.afRf.generator.internalGenerator.multiTone.tone.
↪enable.get(internalGen = repcap.InternalGen.Default, toneNumber = repcap.
↪ToneNumber.Default)
```

Enables or disables a selected tone list entry for multitone generation.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

param toneNumber
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tone')

return
tone_status: OFF | ON

set(*tone_status: bool*, *internalGen=InternalGen.Default*, *toneNumber=ToneNumber.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:MTONe:TONE<no>:ENABle
driver.source.afRf.generator.internalGenerator.multiTone.tone.enable.set(tone_
↪status = False, internalGen = repcap.InternalGen.Default, toneNumber = repcap.
↪ToneNumber.Default)
```

Enables or disables a selected tone list entry for multitone generation.

param tone_status
OFF | ON

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

param toneNumber
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Tone')

6.15.1.1.11.23 Second

class SecondCls
Second commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.second.clone()
```

Subgroups

6.15.1.1.11.24 MultiTone

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator:SECond:MTONe:TLEVel
```

class MultiToneCls
MultiTone commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tlevel() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:SECond:MTONe:TLEVel
value: List[float] = driver.source.afRf.generator.internalGenerator.second.
↪multiTone.get_tlevel()
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

return
tlevel: Range: 0 % to 100 %, Unit: %

set_tlevel(*tlevel: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:SECond:MTONe:TLEVel
driver.source.afRf.generator.internalGenerator.second.multiTone.set_
↪tlevel(tlevel = [1.1, 2.2, 3.3])
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

param tlevel
Range: 0 % to 100 %, Unit: %

6.15.1.1.11.25 Third

class ThirdCls

Third commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.internalGenerator.third.clone()
```

Subgroups

6.15.1.1.11.26 MultiTone

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator:THIRd:MTONe:TLEVel
```

class MultiToneCls

MultiTone commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_tlevel() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:THIRd:MTONe:TLEVel
value: List[float] = driver.source.afRf.generator.internalGenerator.third.
↪multiTone.get_tlevel()
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

return
tlevel: Range: 0 % to 100 %, Unit: %

set_tlevel(*tlevel: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator:THIRd:MTONe:TLEVel
driver.source.afRf.generator.internalGenerator.third.multiTone.set_
↪tlevel(tlevel = [1.1, 2.2, 3.3])
```

Sets the total level for the multitone audio generator 1, 2, 3 or 4.

param tlevel
Range: 0 % to 100 %, Unit: %

6.15.1.1.11.27 Tmode

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:TMODe
```

class TmodeCls

Tmode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*internalGen=InternalGen.Default*) → ToneTypeA

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:TMODe
value: enums.ToneTypeA = driver.source.afRf.generator.internalGenerator.tmode.
↪get(internalGen = repcap.InternalGen.Default)
```

Selects the tone mode of an internal audio generator.

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

return
tone_type: STONe | DTONe | MTONe | NOISe | DTMF | SELCall | FDIaling | SCAL | SQUare STONe Single-tone signal DTONe Dual-tone signal MTONe Multitone signal NOISe Noise signal DTMF DTMF sequence SELCall SelCall selective calling FDIaling Free dialing SCAL SELCAL selective calling SQUare Square signal

set(*tone_type: ToneTypeA*, *internalGen=InternalGen.Default*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:IGENerator<nr>:TMODe
driver.source.afRf.generator.internalGenerator.tmode.set(tone_type = enums.
↪ToneTypeA.DTMF, internalGen = repcap.InternalGen.Default)
```

Selects the tone mode of an internal audio generator.

param tone_type
STONe | DTONe | MTONe | NOISe | DTMF | SELCall | FDIaling | SCAL | SQUare STONe Single-tone signal DTONe Dual-tone signal MTONe Multitone signal NOISe Noise signal DTMF DTMF sequence SELCall SelCall selective calling FDIaling Free dialing SCAL SELCAL selective calling SQUare Square signal

param internalGen
optional repeated capability selector. Default value: Nr1 (settable in the interface 'InternalGenerator')

6.15.1.1.12 Modulator

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:MODulator:FDEViation
SOURce:AFRF:GENerator<Instance>:MODulator:PDEViation
SOURce:AFRF:GENerator<Instance>:MODulator:MDEPth
SOURce:AFRF:GENerator<Instance>:MODulator
```

class ModulatorCls

Modulator commands group definition. 9 total commands, 2 Subgroups, 4 group commands

get_fdeviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FDEViation
value: float = driver.source.afRf.generator.modulator.get_fdeviation()
```

Specifies the maximum frequency deviation for the FM modulation scheme.

return
freq_deviation: Range: 0 Hz to 100 kHz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:MDEPth
value: float = driver.source.afRf.generator.modulator.get_mod_depth()
```

Specifies the modulation depth for the AM modulation scheme.

return
modulation_depth: Range: 0 % to 100 %, Unit: %

get_pdeviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:PDEViation
value: float = driver.source.afRf.generator.modulator.get_pdeviation()
```

Specifies the maximum phase deviation for the PM modulation scheme.

return
phase_deviation: Range: 0 rad to 10 rad, Unit: rad

get_value() → SignalSource

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator
value: enums.SignalSource = driver.source.afRf.generator.modulator.get_value()
```

Selects the source of an audio signal to be transported via the RF carrier.

return
modulator_source: GEN3 | GEN4 | GENB | AFI1 | AFI2 | AFIB | SPIL | SPIR | SPIN
GEN3 Audio generator 3 GEN4 Audio generator 4 GENB Audio generator 3 and 4 AFI1 AF1 IN AFI2 AF2 IN AFIB AF1 IN and AF2 IN SPIL SPDIF IN, left channel SPIR SPDIF IN, right channel SPIN SPDIF IN, both channels

set_fdeviation(*freq_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FDEViation
driver.source.afRf.generator.modulator.set_fdeviation(freq_deviation = 1.0)
```

Specifies the maximum frequency deviation for the FM modulation scheme.

param freq_deviation
Range: 0 Hz to 100 kHz, Unit: Hz

set_mod_depth(*modulation_depth: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:MDEPth
driver.source.afRf.generator.modulator.set_mod_depth(modulation_depth = 1.0)
```

Specifies the modulation depth for the AM modulation scheme.

param modulation_depth
Range: 0 % to 100 %, Unit: %

set_pdeviation(*phase_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:PDEViation
driver.source.afRf.generator.modulator.set_pdeviation(phase_deviation = 1.0)
```

Specifies the maximum phase deviation for the PM modulation scheme.

param phase_deviation
Range: 0 rad to 10 rad, Unit: rad

set_value(*modulator_source: SignalSource*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator
driver.source.afRf.generator.modulator.set_value(modulator_source = enums.
↪SignalSource.AFI1)
```

Selects the source of an audio signal to be transported via the RF carrier.

param modulator_source
GEN3 | GEN4 | GENB | AFI1 | AFI2 | AFIB | SPIL | SPIR | SPIN GEN3 Audio generator 3 GEN4 Audio generator 4 GENB Audio generator 3 and 4 AFI1 AF1 IN AFI2 AF2 IN AFIB AF1 IN and AF2 IN SPIL SPDIF IN, left channel SPIR SPDIF IN, right channel SPIN SPDIF IN, both channels

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.modulator.clone()
```

Subgroups

6.15.1.1.12.1 Enable

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:MODulator:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Left: bool: OFF | ON
- Right: bool: OFF | ON

get() → EnableStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:ENABle
value: EnableStruct = driver.source.afRf.generator.modulator.enable.get()
```

Enables or disables the audio signal input paths of the modulator. For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only <Left> is relevant. <Right> has no effect.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*left: bool*, *right: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:ENABle
driver.source.afRf.generator.modulator.enable.set(left = False, right = False)
```

Enables or disables the audio signal input paths of the modulator. For FM stereo, the settings configure the left and the right audio channel. For other modulation types, only <Left> is relevant. <Right> has no effect.

param left
OFF | ON

param right
OFF | ON

6.15.1.1.12.2 FmStereo

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:MDEViation
```

class FmStereoCls

FmStereo commands group definition. 4 total commands, 3 Subgroups, 1 group commands

get_mdeviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:MDEViation
value: float = driver.source.afRf.generator.modulator.fmStereo.get_mdeviation()
```

Queries the frequency deviation of the FM stereo multiplex signal. The value is calculated from the frequency deviations configured for the signal components.

return
max_freq_deviation: Range: 0 Hz to 100 kHz, Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.modulator.fmStereo.clone()
```

Subgroups

6.15.1.1.12.3 Madeviation

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:MADeviation
```

class MadeviationCls

Madeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Max_Audio_Deviation: float: Maximum audio deviation in Hz Range: 0 Hz to 100 kHz, Unit: Hz
- Ratio: float: Maximum audio deviation as percentage of the multiplex deviation Range: 0 % to 100 %, Unit: %

get() → GetStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:MADeviation
value: GetStruct = driver.source.afRf.generator.modulator.fmStereo.madeviation.
↪get()
```

Specifies the maximum frequency deviation for the audio signal component of a generated FM stereo multiplex signal. The allowed range depends on the frequency deviation of the other signal components. The total deviation of the multiplex signal must not exceed 100 kHz. A query returns <MaxAudioDeviation>, <Ratio>.

return
structure: for return value, see the help for GetStruct structure arguments.

set(*max_audio_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:MADeviation
driver.source.afRf.generator.modulator.fmStereo.madeviation.set(max_audio_
↪deviation = 1.0)
```

Specifies the maximum frequency deviation for the audio signal component of a generated FM stereo multiplex signal. The allowed range depends on the frequency deviation of the other signal components. The total deviation of the multiplex signal must not exceed 100 kHz. A query returns <MaxAudioDeviation>, <Ratio>.

param max_audio_deviation
Maximum audio deviation in Hz Range: 0 Hz to 100 kHz, Unit: Hz

6.15.1.1.12.4 Pdeviation

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:PDEViation
```

class PdeviationCls

Pdeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Max_Pilot_Deviation: float: Maximum pilot deviation in Hz Range: 0 Hz to 10 kHz, Unit: Hz
- Ratio: float: Maximum pilot deviation as percentage of the multiplex deviation Range: 0 % to 100 %, Unit: %

get() → GetStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:PDEViation
value: GetStruct = driver.source.afRf.generator.modulator.fmStereo.pdeviation.
↪get()
```

Specifies the maximum frequency deviation for the pilot tone of a generated FM stereo multiplex signal. The allowed range depends on the frequency deviation of the other signal components. The total deviation of the multiplex signal must not exceed 100 kHz. A query returns <MaxPilotDeviation>, <Ratio>.

return
structure: for return value, see the help for GetStruct structure arguments.

set(*max_pilot_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:PDEViation
driver.source.afRf.generator.modulator.fmStereo.pdeviation.set(max_pilot_
↪deviation = 1.0)
```

Specifies the maximum frequency deviation for the pilot tone of a generated FM stereo multiplex signal. The allowed range depends on the frequency deviation of the other signal components. The total deviation of the multiplex signal must not exceed 100 kHz. A query returns <MaxPilotDeviation>, <Ratio>.

param max_pilot_deviation
Maximum pilot deviation in Hz Range: 0 Hz to 10 kHz, Unit: Hz

6.15.1.1.12.5 RdsDeviation

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:RDSDeviation
```

class RdsDeviationCls

RdsDeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rds_Deviation: float: Maximum RDS deviation in Hz Range: 0 Hz to 10 kHz, Unit: Hz
- Ratio: float: Maximum RDS deviation as percentage of the multiplex deviation Range: 0 % to 100 %, Unit: %

get() → GetStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:RDSDeviation
value: GetStruct = driver.source.afRf.generator.modulator.fmStereo.rdsDeviation.
↪get()
```

Specifies the maximum frequency deviation for the RDS signal component of a generated FM stereo multiplex signal. The allowed range depends on the frequency deviation of the other signal components. The total deviation of the multiplex signal must not exceed 100 kHz. A query returns <RDSDeviation>, <Ratio>.

return
structure: for return value, see the help for GetStruct structure arguments.

set(*rds_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:MODulator:FMSTereo:RDSDeviation
driver.source.afRf.generator.modulator.fmStereo.rdsDeviation.set(rds_deviation␣
↪= 1.0)
```

Specifies the maximum frequency deviation for the RDS signal component of a generated FM stereo multiplex signal. The allowed range depends on the frequency deviation of the other signal components. The total deviation of the multiplex signal must not exceed 100 kHz. A query returns <RDSDeviation>, <Ratio>.

param rds_deviation
Maximum RDS deviation in Hz Range: 0 Hz to 10 kHz, Unit: Hz

6.15.1.1.13 Nxdn

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:NXDN:PATTern
SOURce:AFRF:GENerator<Instance>:NXDN:SVALue
SOURce:AFRF:GENerator<Instance>:NXDN:RAN
SOURce:AFRF:GENerator<Instance>:NXDN:SUID
SOURce:AFRF:GENerator<Instance>:NXDN:DUID
```

(continues on next page)

(continued from previous page)

```
SOURce:AFRF:GENerator<Instance>:NXDN:TRANsmission
SOURce:AFRF:GENerator<Instance>:NXDN:MODE
SOURce:AFRF:GENerator<Instance>:NXDN:SDEViation
SOURce:AFRF:GENerator<Instance>:NXDN:SRATe
SOURce:AFRF:GENerator<Instance>:NXDN:FILTer
SOURce:AFRF:GENerator<Instance>:NXDN:ROFactor
```

class NxdnCls

Nxdn commands group definition. 11 total commands, 0 Subgroups, 11 group commands

get_duid() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:DUID
value: str = driver.source.afRf.generator.nxdn.get_duid()
```

Configures the destination ID to be signaled to the DUT, for NXDN.

return
duid: Range: #H0 to #HFFFF

get_filter_py() → FilterNxDn

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:FILTer
value: enums.FilterNxDn = driver.source.afRf.generator.nxdn.get_filter_py()
```

Queries the filter type used for pulse shaping for NXDN.

return
filter_py: NXTX

get_mode() → FskMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:MODE
value: enums.FskMode = driver.source.afRf.generator.nxdn.get_mode()
```

Queries the modulation type used for NXDN.

return
mode: FSK4

get_pattern() → NxdnPattern

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:PATTern
value: enums.NxdnPattern = driver.source.afRf.generator.nxdn.get_pattern()
```

Selects the bit pattern to be transmitted as payload for NXDN.

return
pattern: P1031 | P1011 | SILence | PRBS9 | PRBS15 | RSYR | RLD | RAW1 | RAW2 | RA1 | RA0 | R10A | RPRB9 | RPRB15

get_ran() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:RAN
value: str = driver.source.afRf.generator.nxdn.get_ran()
```

Configures the radio access number to be signaled to the DUT, for NXDN.

return
ran: Range: #H0 to #H3F

get_ro_factor() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:ROFactor
value: float = driver.source.afRf.generator.nxdn.get_ro_factor()
```

Queries the roll-off factor of the filter used for pulse shaping for NXDN.

return
ro_factor: Range: 0.2 to 0.2

get_standard_dev() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:SDEViation
value: List[float] = driver.source.afRf.generator.nxdn.get_standard_dev()
```

Queries the frequency deviations used for 4FSK modulation, for NXDN.

return
sdeviation: List of four frequency deviations, for the symbols 01, 00, 10, 11. Range: -3000 Hz to 3000 Hz, Unit: Hz

get_suid() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:SUID
value: str = driver.source.afRf.generator.nxdn.get_suid()
```

Configures the sender ID to be signaled to the DUT, for NXDN.

return
suid: Range: #H0 to #HFFFF

get_svalue() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:SVALue
value: str = driver.source.afRf.generator.nxdn.get_svalue()
```

Specifies the seed value for the PRBS generator, for NXDN.

return
svalue: Range: #H0 to #H1FF

get_symbol_rate() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:SRATe
value: float = driver.source.afRf.generator.nxdn.get_symbol_rate()
```

Queries the symbol rate resulting from the configured transmission mode, for NXDN.

return
srate: Range: 0 symbol/s to 100E+6 symbol/s, Unit: symbol/s

get_transmission() → Transmission

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:TRANsmission
value: enums.Transmission = driver.source.afRf.generator.nxdn.get_transmission()
```

Selects the transmission mode for NXDN.

return
trans: EHR4800 | EHR9600 | EFR9600 Enhanced half-rate (EHR) or full-rate (EFR) speech codec Transmission rate 4800 bps or 9600 bps

set_duid(*duid: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:DUID
driver.source.afRf.generator.nxdn.set_duid(duid = r1)
```

Configures the destination ID to be signaled to the DUT, for NXDN.

param duid
Range: #H0 to #HFFFF

set_pattern(*pattern: NxdnPattern*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:PATTern
driver.source.afRf.generator.nxdn.set_pattern(pattern = enums.NxdnPattern.P1011)
```

Selects the bit pattern to be transmitted as payload for NXDN.

param pattern
P1031 | P1011 | SILence | PRBS9 | PRBS15 | RSYR | RLD | RAW1 | RAW2 | RA1 | RA0 | R10A | RPRB9 | RPRB15

set_ran(*ran: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:RAN
driver.source.afRf.generator.nxdn.set_ran(ran = r1)
```

Configures the radio access number to be signaled to the DUT, for NXDN.

param ran
Range: #H0 to #H3F

set_suid(*suid: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:SUID
driver.source.afRf.generator.nxdn.set_suid(suid = r1)
```

Configures the sender ID to be signaled to the DUT, for NXDN.

param suid
Range: #H0 to #HFFFF

set_svalue(*svalue: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:SVALue
driver.source.afRf.generator.nxdn.set_svalue(svalue = r1)
```

Specifies the seed value for the PRBS generator, for NXDN.

param svalue
Range: #H0 to #H1FF

set_transmission(*trans: Transmission*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:NXDN:TRANsmission
driver.source.afRf.generator.nxdn.set_transmission(trans = enums.Transmission.
↪EFR9600)
```

Selects the transmission mode for NXDN.

param trans
EHR4800 | EHR9600 | EFR9600 Enhanced half-rate (EHR) or full-rate (EFR) speech codec Transmission rate 4800 bps or 9600 bps

6.15.1.1.14 Pocsag

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:POCSag:REPetition
SOURce:AFRF:GENerator<Instance>:POCSag:SDEViation
SOURce:AFRF:GENerator<Instance>:POCSag:IMODulation
SOURce:AFRF:GENerator<Instance>:POCSag:SRATe
SOURce:AFRF:GENerator<Instance>:POCSag:PADDress
SOURce:AFRF:GENerator<Instance>:POCSag:FBITs
SOURce:AFRF:GENerator<Instance>:POCSag:PTYPe
SOURce:AFRF:GENerator<Instance>:POCSag:MESSage
SOURce:AFRF:GENerator<Instance>:POCSag:MODE
```

class PocsagCls

Pocsag commands group definition. 9 total commands, 0 Subgroups, 9 group commands

get_fbits() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:FBITs
value: str = driver.source.afRf.generator.pocsag.get_fbits()
```

Configures the function bits for POCSAG.

return
fbits: Range: #B0 to #B11

get_imodulation() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:IMODulation
value: bool = driver.source.afRf.generator.pocsag.get_imodulation()
```

Enables inverted modulation (symbol 0 negative deviation, symbol 1 positive deviation) , for POCSAG.

return
imod: OFF | ON

get_message() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:MESSage
value: str = driver.source.afRf.generator.pocsag.get_message()
```

Specifies a character sequence for numeric and alphanumeric messages, for POCSAG.

return
content: Message as string

get_mode() → FskMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:MODE
value: enums.FskMode = driver.source.afRf.generator.pocsag.get_mode()
```

Queries the modulation type used for POCSAG.

return
mode: FSK2

get_paddress() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:PADDress
value: int = driver.source.afRf.generator.pocsag.get_paddress()
```

Configures the pager address to which a POCSAG transmission is sent.

return
paddress: Range: 0 to 2097151

get_ptype() → PagerType

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:PTYPe
value: enums.PagerType = driver.source.afRf.generator.pocsag.get_ptype()
```

Specifies whether a message is transmitted to the DUT and which message format is used, for POCSAG.

return
pager_type: NUMeric | ALPHanumeric | TONLy NUMeric: message in numeric format ALPHanumeric: message in alpha-numeric format TONLy: no message

get_repetition() → RepeatMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:REPetition
value: enums.RepeatMode = driver.source.afRf.generator.pocsag.get_repetition()
```

Specifies how often the bit sequence is processed for the POCSAG standard.

return
repetition: CONTinuous | SINGle SINGle: Single transmission of the bit sequence CONTinuous: Continuous repetition of the bit sequence

get_standard_dev() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:SDEViation
value: List[float] = driver.source.afRf.generator.pocsag.get_standard_dev()
```

Configures the frequency deviations used for 2FSK modulation, for POCSAG. The values apply if inverted modulation is disabled. A query returns <DeviationS0>, <DeviationS1>.

return
sdeviation: No help available

get_symbol_rate() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:SRATe
value: int = driver.source.afRf.generator.pocsag.get_symbol_rate()
```

Configures the symbol rate for POCSAG.

return
srate: Range: 0 symbol/s to 5000 symbol/s, Unit: symbol/s

set_fbits(*fbits: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:FBITs
driver.source.afRf.generator.pocsag.set_fbits(fbits = r1)
```

Configures the function bits for POCSAG.

param fbits
Range: #B0 to #B11

set_imodulation(*imod: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:IMODulation
driver.source.afRf.generator.pocsag.set_imodulation(imod = False)
```

Enables inverted modulation (symbol 0 negative deviation, symbol 1 positive deviation) , for POCSAG.

param imod
OFF | ON

set_message(*content: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:MESSage
driver.source.afRf.generator.pocsag.set_message(content = '1')
```

Specifies a character sequence for numeric and alphanumeric messages, for POCSAG.

param content
Message as string

set_paddress(*paddress: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:PADDress
driver.source.afRf.generator.pocsag.set_paddress(paddress = 1)
```

Configures the pager address to which a POCSAG transmission is sent.

param paddress
Range: 0 to 2097151

set_ptype(*pager_type: PagerType*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:PTYPe
driver.source.afRf.generator.pocsag.set_ptype(pager_type = enums.PagerType.
↪ALPHanumeric)
```

Specifies whether a message is transmitted to the DUT and which message format is used, for POCSAG.

param pager_type
NUMeric | ALPHanumeric | TONLy NUMeric: message in numeric format ALPHanumeric: message in alpha-numeric format TONLy: no message

set_repetition(*repetition: RepeatMode*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:REPetition
driver.source.afRf.generator.pocsag.set_repetition(repetition = enums.
↪RepeatMode.CONTinuous)
```

Specifies how often the bit sequence is processed for the POCSAG standard.

param repetition
CONTinuous | SINGle SINGle: Single transmission of the bit sequence CONTinuous: Continuous repetition of the bit sequence

set_standard_dev(*sdeviation: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:SDEViation
driver.source.afRf.generator.pocsag.set_standard_dev(sdeviation = [1.1, 2.2, 3.
↪3])
```

Configures the frequency deviations used for 2FSK modulation, for POCSAG. The values apply if inverted modulation is disabled. A query returns <DeviationS0>, <DeviationS1>.

param sdeviation
No help available

set_symbol_rate(*srate: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:POCSag:SRATe
driver.source.afRf.generator.pocsag.set_symbol_rate(srate = 1)
```

Configures the symbol rate for POCSAG.

param srate
Range: 0 symbol/s to 5000 symbol/s, Unit: symbol/s

6.15.1.1.15 PtFive

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:PTFive:EMERgency
SOURce:AFRF:GENerator<Instance>:PTFive:SID
SOURce:AFRF:GENerator<Instance>:PTFive:TGID
SOURce:AFRF:GENerator<Instance>:PTFive:NAC
SOURce:AFRF:GENerator<Instance>:PTFive:PATTern
SOURce:AFRF:GENerator<Instance>:PTFive:MODE
```

class PtFiveCls

PtFive commands group definition. 10 total commands, 1 Subgroups, 6 group commands

get_emergency() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:EMERgency
value: bool = driver.source.afRf.generator.ptFive.get_emergency()
```

Configures the emergency bit to be signaled to the DUT, for P25.

return
emergency: OFF | ON

get_mode() → P25Mode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:MODE
value: enums.P25Mode = driver.source.afRf.generator.ptFive.get_mode()
```

Specifies the modulation type used for P25 phase 1 modulation.

return
mode: C4FM

get_nac() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:NAC
value: str = driver.source.afRf.generator.ptFive.get_nac()
```

Configures the network access code to be signaled to the DUT.

return
nac: Range: #H0 to #HFFF

get_pattern() → P25Pattern

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:PATTern
value: enums.P25Pattern = driver.source.afRf.generator.ptFive.get_pattern()
```

Selects the bit pattern to be transmitted as payload for P25.

return
pattern: P1011 | SILence | INTerference | BUSY | IDLE | CALibration | RSYR | RLD | C4FM | RAW1 | RA1 | RA0 | R10A | RPRB9 | RPRB15

get_sid() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:SID
value: float = driver.source.afRf.generator.ptFive.get_sid()
```

Configures the source ID to be signaled to the DUT.

return
source_id: Range: #H0 to #HFFFFFF

get_tgid() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:TGID
value: float = driver.source.afRf.generator.ptFive.get_tgid()
```

Configures the talk group ID to be signaled to the DUT.

return
tgroup_id: Range: #H0 to #HFFFF

set_emergency(*emergency: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:EMERgency
driver.source.afRf.generator.ptFive.set_emergency(emergency = False)
```

Configures the emergency bit to be signaled to the DUT, for P25.

param emergency
OFF | ON

set_mode(*mode: P25Mode*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:MODE
driver.source.afRf.generator.ptFive.set_mode(mode = enums.P25Mode.C4FM)
```

Specifies the modulation type used for P25 phase 1 modulation.

param mode
C4FM

set_nac(*nac: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:NAC
driver.source.afRf.generator.ptFive.set_nac(nac = r1)
```

Configures the network access code to be signaled to the DUT.

param nac
Range: #H0 to #HFFF

set_pattern(*pattern: P25Pattern*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:PATTern
driver.source.afRf.generator.ptFive.set_pattern(pattern = enums.P25Pattern.BUSY)
```

Selects the bit pattern to be transmitted as payload for P25.

param pattern
P1011 | SILence | INTerference | BUSY | IDLE | CALibration | RSYR | RLD | C4FM | RAW1 | RA1 | RA0 | R10A | RPRB9 | RPRB15

set_sid(*source_id: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:SID
driver.source.afRf.generator.ptFive.set_sid(source_id = 1.0)
```

Configures the source ID to be signaled to the DUT.

param source_id
Range: #H0 to #HFFFFFF

set_tgid(*tgroup_id: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:TGID
driver.source.afRf.generator.ptFive.set_tgid(tgroup_id = 1.0)
```

Configures the talk group ID to be signaled to the DUT.

param tgroup_id
Range: #H0 to #HFFFF

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.ptFive.clone()
```

Subgroups

6.15.1.1.15.1 CfFm

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:SRATe
SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:FILTer
SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:ROFactor
SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:SDEViation
```

class CfFmCls

CfFm commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_filter_py() → PtFiveFilter

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:FILTer
value: enums.PtFiveFilter = driver.source.afRf.generator.ptFive.cfFm.get_filter_
↪py()
```

Queries the filter type used for pulse shaping for P25 with C4FM modulation.

return
filter_py: C4FM | RC

get_ro_factor() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:ROFactor
value: float = driver.source.afRf.generator.ptFive.cfFm.get_ro_factor()
```

Queries the roll-off factor of the filter used for pulse shaping for P25 with C4FM modulation.

return
ro_factor: Range: 0.2 to 0.2

get_standard_dev() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:SDEViation
value: List[float] = driver.source.afRf.generator.ptFive.cfFm.get_standard_dev()
```

Queries the deviations used for C4FM modulation, for P25.

return
sdeviation: List of four deviations, for the symbols 01, 00, 10, 11. Range: -1800 Hz to 1800 Hz, Unit: Hz

get_symbol_rate() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:PTFive:CFFM:SRATe
value: float = driver.source.afRf.generator.ptFive.cfFm.get_symbol_rate()
```

Queries the symbol rate for P25 with C4FM modulation.

return
srate: Range: 4800 symbol/s to 4800 symbol/s , Unit: symbol/s

6.15.1.1.16 Reliability

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:RELiability
SOURce:AFRF:GENerator<Instance>:RELiability:ALL
```

class ReliabilityCls

Reliability commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class AllStruct

Structure for reading output parameters. Fields:

- Reliability: int: See 'Reliability indicator values'
- Reliability_Msg: str: String indicating the error reason If there is no error (reliability value = 0) , the string is empty.
- Reliability_Add_Info: str: String providing additional information for an error If there is no error (reliability value = 0) , the string is empty.

get(*details: str = None*) → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RELiability
value: str = driver.source.afRf.generator.reliability.get(details = '1')
```

Queries whether the generator has detected an error or not. If you have problems to generate a signal, use this command for troubleshooting. The returned parameters comprise a reliability indicator value and optionally, an error reason.

Use RsCma.reliability.last_value to read the updated reliability indicator.

param details
To return an error reason in addition to the reliability indicator value, append 'Details' to the query: SOUR:AFRF:GEN:REL? 'DETails'

return
reliability_msg: String indicating the error reason If there is no error (reliability value = 0) , the string is empty. The parameter is only returned if parameter Details = 'DETails'.

get_all() → AllStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RELiability:ALL
value: AllStruct = driver.source.afRf.generator.reliability.get_all()
```

Queries whether the generator has detected an error or not. If you have problems to generate a signal, use this command for troubleshooting. The returned parameters comprise a reliability indicator value, an error reason and an additional information string.

return
structure: for return value, see the help for AllStruct structure arguments.

6.15.1.1.17 RfSettings

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:RFSettings:DGAin
SOURce:AFRF:GENerator<Instance>:RFSettings:EATTenuation
SOURce:AFRF:GENerator<Instance>:RFSettings:FREQuency
SOURce:AFRF:GENerator<Instance>:RFSettings:LEVel
SOURce:AFRF:GENerator<Instance>:RFSettings:PEPower
SOURce:AFRF:GENerator<Instance>:RFSettings:RFCoupling
SOURce:AFRF:GENerator<Instance>:RFSettings:CONNector
SOURce:AFRF:GENerator<Instance>:RFSettings:CHANnel
SOURce:AFRF:GENerator<Instance>:RFSettings:COFFset
```

class RfSettingsCls

RfSettings commands group definition. 11 total commands, 2 Subgroups, 9 group commands

get_channel() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:CHANnel
value: int = driver.source.afRf.generator.rfSettings.get_channel()
```

Specifies the center frequency of the unmodulated RF carrier via a channel number, according to the configured channel definition.

return
channel: Range: 0 Ch to 9999 Ch, Unit: Ch

get_coffset() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:COFFset
value: float = driver.source.afRf.generator.rfSettings.get_coffset()
```

Shifts the center frequency of the unmodulated RF carrier by a channel offset, relative to the frequency defined via the channel number. The range depends on the channel spacing, defined via method RsCma.Source.AfRf.Generator.Cdefinition. cspace.

return
channel_offset: Range: -Spacing/2 Hz to +Spacing/2 Hz, Unit: Hz

get_connector() → OutputConnector

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:CONNector
value: enums.OutputConnector = driver.source.afRf.generator.rfSettings.get_
↪connector()
```

Selects the output connector for the generated RF signal.

return
output_connector: RFCom | RFOut

get_dgain() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:DGAin
value: float = driver.source.afRf.generator.rfSettings.get_dgain()
```

Specifies a digital gain and thus modifies the configured RMS base level by a specific value.

return
dig_gain: Range: -30 dB to 0 dB, Unit: dB

get_eattenuation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:EATTenuation
value: float = driver.source.afRf.generator.rfSettings.get_eattenuation()
```

Specifies the external attenuation in the RF output path. Negative values specify a gain.

return
rf_output_ext_att: Range: -50 dB to 90 dB, Unit: dB

get_frequency() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:FREQuency
value: float = driver.source.afRf.generator.rfSettings.get_frequency()
```

Specifies the center frequency of the unmodulated RF carrier.

return
frequency: Range: 100 kHz to 3 GHz, Unit: Hz

get_level() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:LEVel
value: float = driver.source.afRf.generator.rfSettings.get_level()
```

Specifies the RMS level of the unmodulated RF signal. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

return
level: Unit: dBm

get_pe_power() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:PEPower
value: float = driver.source.afRf.generator.rfSettings.get_pe_power()
```

Queries the peak envelope power (PEP) .

return
peak_envelope_power: Unit: dBm

get_rf_coupling() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:RFCoupling
value: bool = driver.source.afRf.generator.rfSettings.get_rf_coupling()
```

No command help available

return
enable: No help available

set_channel(*channel: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:CHANnel
driver.source.afRf.generator.rfSettings.set_channel(channel = 1)
```

Specifies the center frequency of the unmodulated RF carrier via a channel number, according to the configured channel definition.

param channel
Range: 0 Ch to 9999 Ch, Unit: Ch

set_coffset(*channel_offset: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:COFFset
driver.source.afRf.generator.rfSettings.set_coffset(channel_offset = 1.0)
```

Shifts the center frequency of the unmodulated RF carrier by a channel offset, relative to the frequency defined via the channel number. The range depends on the channel spacing, defined via method RsCma.Source.AfRf.Generator.Cdefinition. cspace.

param channel_offset
Range: -Spacing/2 Hz to +Spacing/2 Hz, Unit: Hz

set_connector(*output_connector: OutputConnector*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:CONNector
driver.source.afRf.generator.rfSettings.set_connector(output_connector = enums.
↪OutputConnector.RFCom)
```

Selects the output connector for the generated RF signal.

param output_connector
RFCom | RFOut

set_dgain(*dig_gain: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:DGAin
driver.source.afRf.generator.rfSettings.set_dgain(dig_gain = 1.0)
```

Specifies a digital gain and thus modifies the configured RMS base level by a specific value.

param dig_gain
Range: -30 dB to 0 dB, Unit: dB

set_eattenuation(*rf_output_ext_att: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:EATTenuation
driver.source.afRf.generator.rfSettings.set_eattenuation(rf_output_ext_att = 1.
↪0)
```

Specifies the external attenuation in the RF output path. Negative values specify a gain.

param rf_output_ext_att
Range: -50 dB to 90 dB, Unit: dB

set_frequency(*frequency: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:FREQuency
driver.source.afRf.generator.rfSettings.set_frequency(frequency = 1.0)
```

Specifies the center frequency of the unmodulated RF carrier.

param frequency
Range: 100 kHz to 3 GHz, Unit: Hz

set_level(*level: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:LEVel
driver.source.afRf.generator.rfSettings.set_level(level = 1.0)
```

Specifies the RMS level of the unmodulated RF signal. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

param level
Unit: dBm

set_rf_coupling(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:RFCoupling
driver.source.afRf.generator.rfSettings.set_rf_coupling(enable = False)
```

No command help available

param enable
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.rfSettings.clone()
```

Subgroups

6.15.1.1.17.1 FarFrequency

class FarFrequencyCls

FarFrequency commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.rfSettings.farFrequency.clone()
```

Subgroups

6.15.1.1.17.2 Action

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:RFSettings:FARFrequency:ACTion
```

class ActionCls

Action commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:FARFrequency:ACTion
driver.source.afRf.generator.rfSettings.farFrequency.action.set()
```

Sets the reference frequency of the channel definition to the current RF carrier center frequency.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:FARFrequency:ACTion
driver.source.afRf.generator.rfSettings.farFrequency.action.set_with_opc()
```

Sets the reference frequency of the channel definition to the current RF carrier center frequency.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.15.1.1.17.3 Rf

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:RFSettings:RF:ENABle
```

class RfCls

Rf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:RF:ENABle
value: bool = driver.source.afRf.generator.rfSettings.rf.get_enable()
```

Enables or disables the RF signal, without changing the generator state.

return
rf_enable: OFF | ON

set_enable(*rf_enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:RFSettings:RF:ENABle
driver.source.afRf.generator.rfSettings.rf.set_enable(rf_enable = False)
```

Enables or disables the RF signal, without changing the generator state.

param rf_enable
OFF | ON

6.15.1.1.18 Sout

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:SOUT
```

class SoutCls

Sout commands group definition. 3 total commands, 2 Subgroups, 1 group commands

class SoutStruct

Response structure. Fields:

- Source_Left: enums.SignalSource: GEN3 | AFI1 | SPIL GEN3 Audio generator 3 AFI1 AF1 IN SPIL SPDIF IN, left channel
- Source_Right: enums.SignalSource: GEN4 | AFI2 | SPIR GEN4 Audio generator 4 AFI2 AF2 IN SPIR SPDIF IN, right channel

get() → SoutStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:SOUT
value: SoutStruct = driver.source.afRf.generator.sout.get()
```

Selects audio signal sources for the left and right channel of the SPDIF OUT connector.

return
structure: for return value, see the help for SoutStruct structure arguments.

set(*source_left: SignalSource*, *source_right: SignalSource*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:SOUT
driver.source.afRf.generator.sout.set(source_left = enums.SignalSource.AFI1,␣
↪source_right = enums.SignalSource.AFI1)
```

Selects audio signal sources for the left and right channel of the SPDIF OUT connector.

param source_left
GEN3 | AFI1 | SPIL GEN3 Audio generator 3 AFI1 AF1 IN SPIL SPDIF IN, left channel

param source_right
GEN4 | AFI2 | SPIR GEN4 Audio generator 4 AFI2 AF2 IN SPIR SPDIF IN, right channel

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.sout.clone()
```

Subgroups

6.15.1.1.18.1 Enable

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:SOUT:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnableStruct

Response structure. Fields:

- Left: bool: OFF | ON
- Right: bool: OFF | ON

get() → EnableStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:SOUT:ENABle
value: EnableStruct = driver.source.afRf.generator.sout.enable.get()
```

Enables or disables the left and right channel of the SPDIF OUT connector.

return
structure: for return value, see the help for EnableStruct structure arguments.

set(*left: bool*, *right: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:SOUT:ENABle
driver.source.afRf.generator.sout.enable.set(left = False, right = False)
```

Enables or disables the left and right channel of the SPDIF OUT connector.

param left
OFF | ON

param right
OFF | ON

6.15.1.1.18.2 Level

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:SOUT:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LevelStruct

Response structure. Fields:

- Level_Left: float: Level for the left channel Range: 0.01 % to 100 %, Unit: %
- Level_Right: float: Level for the right channel Range: 0.01 % to 100 %, Unit: %

get() → LevelStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:SOUT:LEVel
value: LevelStruct = driver.source.afRf.generator.sout.level.get()
```

Specifies the output levels for the SPDIF OUT connector. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

return
structure: for return value, see the help for LevelStruct structure arguments.

set(*level_left: float*, *level_right: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:SOUT:LEVel
driver.source.afRf.generator.sout.level.set(level_left = 1.0, level_right = 1.0)
```

Specifies the output levels for the SPDIF OUT connector. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

param level_left
Level for the left channel Range: 0.01 % to 100 %, Unit: %

param level_right
Level for the right channel Range: 0.01 % to 100 %, Unit: %

6.15.1.1.19 State

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:STATe
SOURce:AFRF:GENerator<Instance>:STATe:ALL
```

class StateCls

State commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get() → GeneratorState

```
# SCPI: SOURce:AFRF:GENerator<Instance>:STATe
value: enums.GeneratorState = driver.source.afRf.generator.state.get()
```

Starts or stops the AFRF generator.

return
gen_state: OFF | ON | PENDing OFF Generator is off ON Generator is running PENDing Start or stop of generator is ongoing

get_all() → List[GeneratorState]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:STATe:ALL
value: List[enums.GeneratorState] = driver.source.afRf.generator.state.get_all()
```

No command help available

return
all_states: No help available

set(*gen_control: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:STATe
driver.source.afRf.generator.state.set(gen_control = False)
```

Starts or stops the AFRF generator.

param gen_control
ON | OFF ON Starts the generator OFF Stops the generator

6.15.1.1.20 Tones

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:TONes:FDEViation
SOURce:AFRF:GENerator<Instance>:TONes:PDEViation
SOURce:AFRF:GENerator<Instance>:TONes:MDEPth
SOURce:AFRF:GENerator<Instance>:TONes
```

class TonesCls

Tones commands group definition. 16 total commands, 3 Subgroups, 4 group commands

get_fdeviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:FDEViation
value: float = driver.source.afRf.generator.tones.get_fdeviation()
```

Specifies the maximum frequency deviation, used in FM mode to add a tone to the RF carrier.

return
freq_deviation: Range: 0 Hz to 10 kHz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:MDEPth
value: float = driver.source.afRf.generator.tones.get_mod_depth()
```

Specifies the modulation depth, used in AM mode to add a tone to the RF carrier.

return
modulation_depth: Range: 0 % to 100 %, Unit: %

get_pdeviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:PDEViation
value: float = driver.source.afRf.generator.tones.get_pdeviation()
```

Specifies the maximum phase deviation, used in PM mode to add a tone to the RF carrier.

return
phase_deviation: Range: 0 rad to 10 rad, Unit: rad

get_value() → ToneTypeB

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes
value: enums.ToneTypeB = driver.source.afRf.generator.tones.get_value()
```

Selects the type of additional tones to be generated.

return
tone_type: NONE | SUBTone | CTCSs | DCS NONE No additional tones SUBTone Single subtone CTCSs Single CTCSS subaudible tone DCS DCS signal

set_fdeviation(*freq_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:FDEViation
driver.source.afRf.generator.tones.set_fdeviation(freq_deviation = 1.0)
```

Specifies the maximum frequency deviation, used in FM mode to add a tone to the RF carrier.

param freq_deviation
Range: 0 Hz to 10 kHz, Unit: Hz

set_mod_depth(*modulation_depth: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:MDEPth
driver.source.afRf.generator.tones.set_mod_depth(modulation_depth = 1.0)
```

Specifies the modulation depth, used in AM mode to add a tone to the RF carrier.

param modulation_depth
Range: 0 % to 100 %, Unit: %

set_pdeviation(*phase_deviation: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:PDEViation
driver.source.afRf.generator.tones.set_pdeviation(phase_deviation = 1.0)
```

Specifies the maximum phase deviation, used in PM mode to add a tone to the RF carrier.

param phase_deviation
Range: 0 rad to 10 rad, Unit: rad

set_value(*tone_type: ToneTypeB*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes
driver.source.afRf.generator.tones.set_value(tone_type = enums.ToneTypeB.CTCSs)
```

Selects the type of additional tones to be generated.

param tone_type
NONE | SUBTone | CTCSs | DCS NONE No additional tones SUBTone Single subtone CTCSs Single CTCSS subaudible tone DCS DCS signal

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.tones.clone()
```

Subgroups

6.15.1.1.20.1 CtCss

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:TONes:CTCSs:ENABle
SOURce:AFRF:GENerator<Instance>:TONes:CTCSs:TNUMber
```

class CtCssCls

CtCss commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:CTCSs:ENABle
value: bool = driver.source.afRf.generator.tones.ctCss.get_enable()
```

Enables or disables the CTCSS tone.

return
enable: OFF | ON

get_tnumber() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:CTCSs:TNUMber
value: int = driver.source.afRf.generator.tones.ctCss.get_tnumber()
```

Selects a CTCSS tone via its number in the tone list.

return
tone_number: Range: 1 to 50

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:CTCSs:ENABle
driver.source.afRf.generator.tones.ctCss.set_enable(enable = False)
```

Enables or disables the CTCSS tone.

param enable
OFF | ON

set_tnumber(*tone_number: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:CTCSs:TNUMber
driver.source.afRf.generator.tones.ctCss.set_tnumber(tone_number = 1)
```

Selects a CTCSS tone via its number in the tone list.

param tone_number
Range: 1 to 50

6.15.1.1.20.2 Dcs

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:TONes:DCS:CWORd
SOURce:AFRF:GENerator<Instance>:TONes:DCS:ENABle
SOURce:AFRF:GENerator<Instance>:TONes:DCS:FSKDeviation
SOURce:AFRF:GENerator<Instance>:TONes:DCS:DRATe
SOURce:AFRF:GENerator<Instance>:TONes:DCS:DROFfset
SOURce:AFRF:GENerator<Instance>:TONes:DCS:TOCLength
```

class DcsCls

Dcs commands group definition. 8 total commands, 2 Subgroups, 6 group commands

get_cword() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:CWORd
value: str = driver.source.afRf.generator.tones.dcs.get_cword()
```

Specifies the DCS code number.

return
sequence: DCS code number as octal number Not allowed octal numbers are automatically rounded to the closest allowed value. Range: #Q20 to #Q777

get_dr_offset() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:DROFfset
value: float = driver.source.afRf.generator.tones.dcs.get_dr_offset()
```

Modifies the used data rate by defining an offset relative to the nominal data rate of 134.4 Bit/s.

return
roffset: Range: -30 bit/s to 30 bit/s, Unit: bit/s

get_drate() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:DRATe
value: float = driver.source.afRf.generator.tones.dcs.get_drate()
```

Queries the data rate used for DCS bit stream transmission.

return
bitrate: Range: 104.4 bit/s to 164.4 bit/s, Unit: bit/s

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:ENABle
value: bool = driver.source.afRf.generator.tones.dcs.get_enable()
```

Enables or disables the DCS signal.

return
enable: OFF | ON

get_fsk_deviation() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:FSKDeviation
value: float = driver.source.afRf.generator.tones.dcs.get_fsk_deviation()
```

Specifies the frequency deviation used for FSK modulation of the carrier with the DCS bit stream.

return
fsk_dev: Range: 0 Hz to 10 kHz, Unit: Hz

get_toc_length() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:TOCLength
value: float = driver.source.afRf.generator.tones.dcs.get_toc_length()
```

Specifies the duration of turn-off code transmissions.

return
off_length: Range: 0 s to 1 s, Unit: s

set_cword(*sequence: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:CWORd
driver.source.afRf.generator.tones.dcs.set_cword(sequence = r1)
```

Specifies the DCS code number.

param sequence
DCS code number as octal number Not allowed octal numbers are automatically rounded to the closest allowed value. Range: #Q20 to #Q777

set_dr_offset(*roffset: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:DROFfset
driver.source.afRf.generator.tones.dcs.set_dr_offset(roffset = 1.0)
```

Modifies the used data rate by defining an offset relative to the nominal data rate of 134.4 Bit/s.

param roffset
Range: -30 bit/s to 30 bit/s, Unit: bit/s

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:ENABle
driver.source.afRf.generator.tones.dcs.set_enable(enable = False)
```

Enables or disables the DCS signal.

param enable
OFF | ON

set_fsk_deviation(*fsk_dev: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:FSKDeviation
driver.source.afRf.generator.tones.dcs.set_fsk_deviation(fsk_dev = 1.0)
```

Specifies the frequency deviation used for FSK modulation of the carrier with the DCS bit stream.

param fsk_dev
Range: 0 Hz to 10 kHz, Unit: Hz

set_toc_length(*off_length: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:TOCLength
driver.source.afRf.generator.tones.dcs.set_toc_length(off_length = 1.0)
```

Specifies the duration of turn-off code transmissions.

param off_length
Range: 0 s to 1 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.tones.dcs.clone()
```

Subgroups

6.15.1.1.20.3 Ifsk

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:TONes:DCS:IFSK:ENABle
```

class IfskCls

Ifsk commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:IFSK:ENABle
value: bool = driver.source.afRf.generator.tones.dcs.ifsk.get_enable()
```

Enables or disables the inversion of the FSK modulation polarity.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:IFSK:ENABle
driver.source.afRf.generator.tones.dcs.ifsk.set_enable(enable = False)
```

Enables or disables the inversion of the FSK modulation polarity.

param enable
OFF | ON

6.15.1.1.20.4 ToCode

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:TONes:DCS:TOCode:ENABle
```

class ToCodeCls

ToCode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:TOCode:ENABle
value: bool = driver.source.afRf.generator.tones.dcs.toCode.get_enable()
```

Enables or disables turn-off code transmissions.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:DCS:TOCode:ENABle
driver.source.afRf.generator.tones.dcs.toCode.set_enable(enable = False)
```

Enables or disables turn-off code transmissions.

param enable
OFF | ON

6.15.1.1.20.5 Subtone

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:TONes:SUBTone:ENABle
SOURce:AFRF:GENerator<Instance>:TONes:SUBTone:FREQuency
```

class SubtoneCls

Subtone commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:SUBTone:ENABle
value: bool = driver.source.afRf.generator.tones.subtone.get_enable()
```

Enables or disables the subtone.

return
enable: OFF | ON

get_frequency() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:SUBTone:FREQuency
value: float = driver.source.afRf.generator.tones.subtone.get_frequency()
```

Specifies the frequency of a generated subtone.

return
frequency: Range: 0 Hz to 2000 Hz, Unit: Hz

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:SUBTone:ENABle
driver.source.afRf.generator.tones.subtone.set_enable(enable = False)
```

Enables or disables the subtone.

param enable
OFF | ON

set_frequency(*frequency: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:TONes:SUBTone:FREQuency
driver.source.afRf.generator.tones.subtone.set_frequency(frequency = 1.0)
```

Specifies the frequency of a generated subtone.

param frequency
Range: 0 Hz to 2000 Hz, Unit: Hz

6.15.1.1.21 UserDefined

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:UDEFined:ENABle
SOURce:AFRF:GENerator<Instance>:UDEFined:IMODulation
SOURce:AFRF:GENerator<Instance>:UDEFined:REPetition
SOURce:AFRF:GENerator<Instance>:UDEFined:PAUSe
SOURce:AFRF:GENerator<Instance>:UDEFined:BANDwidth
SOURce:AFRF:GENerator<Instance>:UDEFined:ILENgth
SOURce:AFRF:GENerator<Instance>:UDEFined:ROFactor
SOURce:AFRF:GENerator<Instance>:UDEFined:FILTer
SOURce:AFRF:GENerator<Instance>:UDEFined:SLENgth
SOURce:AFRF:GENerator<Instance>:UDEFined:DRATe
SOURce:AFRF:GENerator<Instance>:UDEFined:SVALue
SOURce:AFRF:GENerator<Instance>:UDEFined:PATTern
SOURce:AFRF:GENerator<Instance>:UDEFined:SDEViation
SOURce:AFRF:GENerator<Instance>:UDEFined:MODE
```

class UserDefinedCls

UserDefined commands group definition. 14 total commands, 0 Subgroups, 14 group commands

get_bandwidth() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:BANDwidth
value: float = driver.source.afRf.generator.userDefined.get_bandwidth()
```

Selects the bandwidth of the ‘Gauss’ filter for pulse shaping for the user-defined standard.

return
bandwidth: Range: 1000 Hz to 100000 Hz

get_drate() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:DRATe
value: int = driver.source.afRf.generator.userDefined.get_drate()
```

Specifies the data rate for the user-defined standard.

return
drate: Range: 200 bit/s to 115200 bit/s, Unit: bit/s

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:ENABle
value: bool = driver.source.afRf.generator.userDefined.get_enable()
```

Enables or disables FSK modulation of user-defined data on the RF carrier.

return
undefined_enable: No help available

get_filter_py() → GeneratorFilter

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:FILTer
value: enums.GeneratorFilter = driver.source.afRf.generator.userDefined.get_
↪filter_py()
```

Selects the filter type for pulse shaping for the user-defined standard.

return
filter_py: GAUSs | RRC | RC | COS | SINC GAUSs Gauss filter RRC Root-raised-cosine filter RC Raised-cosine filter COS Cosine filter SINC Sinc filter

get_ilength() → ImpulseLength

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:ILENgth
value: enums.ImpulseLength = driver.source.afRf.generator.userDefined.get_
↪ilength()
```

Selects the impulse length of the filter used for pulse shaping for the user-defined standard.

return
impulse_length: T | T2 | T4 | T6 | T8

get_imodulation() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:IMODulation
value: bool = driver.source.afRf.generator.userDefined.get_imodulation()
```

Inverts the frequency deviation of the symbols for the frequency-shift keying modulation.

return
imod: OFF | ON

get_mode() → FskMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:MODE
value: enums.FskMode = driver.source.afRf.generator.userDefined.get_mode()
```

Sets the frequency-shift keying modulation type for the user-defined generator setting.

return
mode: FSK2 | FSK4

get_pattern() → UserDefPattern

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:PATTern
value: enums.UserDefPattern = driver.source.afRf.generator.userDefined.get_
↪pattern()
```

Selects the bit pattern to be transmitted as payload for the user-defined standard.

return
pattern: PRBS6 | PRBS9

get_pause() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:PAUSe
value: float = driver.source.afRf.generator.userDefined.get_pause()
```

Defines the duration of a pause between two repetitions of a bit sequence for the user-defined standard.

return
pause: Range: 0 ms to 10E+3 ms, Unit: s

get_repetition() → RepeatMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:REPetition
value: enums.RepeatMode = driver.source.afRf.generator.userDefined.get_
↪repetition()
```

Specifies how often the bit sequence is processed for the user-defined standard.

return
repetition: CONTinuous | SINGle SINGle: Single transmission of the bit sequence
CONTinuous: Continuous repetition of the bit sequence

get_ro_factor() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:ROFactor
value: float = driver.source.afRf.generator.userDefined.get_ro_factor()
```

Specifies the roll-off factor of the filter used for pulse shaping for the user-defined standard.

return
ro_factor: Range: 0 to 1

get_slength() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:SLENgth
value: int = driver.source.afRf.generator.userDefined.get_slength()
```

Specifies the length of a single bit sequence for the user-defined standard.

return
slength: Range: 0 bits to 16320 bits, Unit: bits

get_standard_dev() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:SDEViation
value: List[float] = driver.source.afRf.generator.userDefined.get_standard_dev()
```

Defines the frequency deviations for the frequency-shift keying modulation (2FSK, 4FSK) of the user-defined standard. A setting command defines the deviation for symbol 01. The deviations for the other symbols are calculated from the setting. A query returns a comma-separated list of four deviations, for symbol 01, 00, 10, 11.

return
sdeviation: Frequency deviation Range: 0 Hz to 100 kHz, Unit: Hz

get_svalue() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:SVALue
value: str = driver.source.afRf.generator.userDefined.get_svalue()
```

Specifies the seed value for the PRBS generator, for the user-defined standard.

return
svalue: Range: #H0 to #H1FF (for PRBS 6 max. #H3F)

set_bandwidth(*bandwidth: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:BANDwidth
driver.source.afRf.generator.userDefined.set_bandwidth(bandwidth = 1.0)
```

Selects the bandwidth of the 'Gauss' filter for pulse shaping for the user-defined standard.

param bandwidth
Range: 1000 Hz to 100000 Hz

set_drate(*drate: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:DRATe
driver.source.afRf.generator.userDefined.set_drate(drate = 1)
```

Specifies the data rate for the user-defined standard.

param drate
Range: 200 bit/s to 115200 bit/s, Unit: bit/s

set_enable(*undefined_enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:ENABle
driver.source.afRf.generator.userDefined.set_enable(undefined_enable = False)
```

Enables or disables FSK modulation of user-defined data on the RF carrier.

param undefined_enable
OFF | ON

set_filter_py(*filter_py: GeneratorFilter*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:FILTer
driver.source.afRf.generator.userDefined.set_filter_py(filter_py = enums.
↪GeneratorFilter.COS)
```

Selects the filter type for pulse shaping for the user-defined standard.

param filter_py
GAUSs | RRC | RC | COS | SINC GAUSs Gauss filter RRC Root-raised-cosine filter RC Raised-cosine filter COS Cosine filter SINC Sinc filter

set_ilength(*impulse_length: ImpulseLength*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:ILENgth
driver.source.afRf.generator.userDefined.set_ilength(impulse_length = enums.
↪ImpulseLength.T)
```

Selects the impulse length of the filter used for pulse shaping for the user-defined standard.

param impulse_length
T | T2 | T4 | T6 | T8

set_imodulation(*imod: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:IMODulation
driver.source.afRf.generator.userDefined.set_imodulation(imod = False)
```

Inverts the frequency deviation of the symbols for the frequency-shift keying modulation.

param imod
OFF | ON

set_mode(*mode: FskMode*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:MODE
driver.source.afRf.generator.userDefined.set_mode(mode = enums.FskMode.FSK2)
```

Sets the frequency-shift keying modulation type for the user-defined generator setting.

param mode
FSK2 | FSK4

set_pattern(*pattern: UserDefPattern*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:PATTern
driver.source.afRf.generator.userDefined.set_pattern(pattern = enums.
↪UserDefPattern.PRBS6)
```

Selects the bit pattern to be transmitted as payload for the user-defined standard.

param pattern
PRBS6 | PRBS9

set_pause(*pause: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:PAUSe
driver.source.afRf.generator.userDefined.set_pause(pause = 1.0)
```

Defines the duration of a pause between two repetitions of a bit sequence for the user-defined standard.

param pause
Range: 0 ms to 10E+3 ms, Unit: s

set_repetition(*repetition: RepeatMode*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:REPetition
driver.source.afRf.generator.userDefined.set_repetition(repetition = enums.
↪RepeatMode.CONTinuous)
```

Specifies how often the bit sequence is processed for the user-defined standard.

param repetition
CONTinuous | SINGle SINGle: Single transmission of the bit sequence CONTinuous: Continuous repetition of the bit sequence

set_ro_factor(*ro_factor: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:ROFactor
driver.source.afRf.generator.userDefined.set_ro_factor(ro_factor = 1.0)
```

Specifies the roll-off factor of the filter used for pulse shaping for the user-defined standard.

param ro_factor
Range: 0 to 1

set_slength(*slength: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:SLENgth
driver.source.afRf.generator.userDefined.set_slength(slength = 1)
```

Specifies the length of a single bit sequence for the user-defined standard.

param slength
Range: 0 bits to 16320 bits, Unit: bits

set_standard_dev(*sdeviation: List[float]*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:SDEViation
driver.source.afRf.generator.userDefined.set_standard_dev(sdeviation = [1.1, 2.
↪2, 3.3])
```

Defines the frequency deviations for the frequency-shift keying modulation (2FSK, 4FSK) of the user-defined standard. A setting command defines the deviation for symbol 01. The deviations for the other symbols are calculated from the setting. A query returns a comma-separated list of four deviations, for symbol 01, 00, 10, 11.

param sdeviation
Frequency deviation Range: 0 Hz to 100 kHz, Unit: Hz

set_svalue(*svalue: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:UDEFined:SVALue
driver.source.afRf.generator.userDefined.set_svalue(svalue = r1)
```

Specifies the seed value for the PRBS generator, for the user-defined standard.

param svalue
Range: #H0 to #H1FF (for PRBS 6 max. #H3F)

6.15.1.1.22 Voip

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:VOIP:ENABle
SOURce:AFRF:GENerator<Instance>:VOIP:PCODec
SOURce:AFRF:GENerator<Instance>:VOIP:LEVel
SOURce:AFRF:GENerator<Instance>:VOIP:AUDio
SOURce:AFRF:GENerator<Instance>:VOIP:FID
SOURce:AFRF:GENerator<Instance>:VOIP:FREQuency
SOURce:AFRF:GENerator<Instance>:VOIP
```

class VoipCls

Voip commands group definition. 20 total commands, 4 Subgroups, 7 group commands

class FrequencyStruct

Structure for reading output parameters. Fields:

- Freq: float: RF carrier center frequency Unit: Hz
- Chanspac: int: Channel spacing Unit: Hz

get_audio() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:AUDio
value: bool = driver.source.afRf.generator.voip.get_audio()
```

Enables or disables feeding an audio signal into the VoIP path.

return
af_2_vo_ip_enable: OFF | ON

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:ENABle
value: bool = driver.source.afRf.generator.voip.get_enable()
```

Enables or disables VoIP.

return
vo_ip_enable: OFF | ON

get_fid() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:FID
value: float = driver.source.afRf.generator.voip.get_fid()
```

Specifies the frequency ID (FID) configured at the DUT.
INTRO_CMD_HELP: Allowed values are, with n = 0 to 39995:

- 0.100 + n * 0.025
- 0.105 + n * 0.025
- 0.110 + n * 0.025
- 0.115 + n * 0.025

Resulting in: 0.100, 0.105, 0.110, 0.115, 0.125, 0.130, 0.135, 0.140, ..., 999.975, 999.980, 999.985, 999.990

return
freq_id: Frequency ID Not allowed values are rounded to the closest allowed value.
Range: 0.1 to 999.99

get_frequency() → FrequencyStruct

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:FREQuency
value: FrequencyStruct = driver.source.afRf.generator.voip.get_frequency()
```

Queries the RF carrier center frequency and the channel spacing resulting from the configured frequency ID.

return
structure: for return value, see the help for FrequencyStruct structure arguments.

get_level() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:LEVel
value: float = driver.source.afRf.generator.voip.get_level()
```

Specifies the audio output level for the VoIP path. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

return
level: Range: 0.01 % to 100 %, Unit: %

get_pcodec() → VoIpCodec

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:PCODec
value: enums.VoIpCodec = driver.source.afRf.generator.voip.get_pcodec()
```

Queries the type of the pulse code modulation (PCM) codec.

return
vo_ip_codec: ALAW

get_value() → VoIpSource

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP
value: enums.VoIpSource = driver.source.afRf.generator.voip.get_value()
```

Selects an audio signal source for the VoIP path.

return
vo_ip_source: GEN4 | AFI1 | AFI2 GEN4 Audio generator 4

set_audio(*af_2_vo_ip_enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:AUDio
driver.source.afRf.generator.voip.set_audio(af_2_vo_ip_enable = False)
```

Enables or disables feeding an audio signal into the VoIP path.

param af_2_vo_ip_enable
OFF | ON

set_enable(*vo_ip_enable: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:ENABle
driver.source.afRf.generator.voip.set_enable(vo_ip_enable = False)
```

Enables or disables VoIP.

param vo_ip_enable
OFF | ON

set_fid(*freq_id: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:FID
driver.source.afRf.generator.voip.set_fid(freq_id = 1.0)
```

Specifies the frequency ID (FID) configured at the DUT.
INTRO_CMD_HELP: Allowed values are, with n = 0 to 39995:

- 0.100 + n * 0.025
- 0.105 + n * 0.025
- 0.110 + n * 0.025
- 0.115 + n * 0.025

Resulting in: 0.100, 0.105, 0.110, 0.115, 0.125, 0.130, 0.135, 0.140, …, 999.975, 999.980, 999.985, 999.990

param freq_id
Frequency ID Not allowed values are rounded to the closest allowed value. Range: 0.1 to 999.99

set_level(*level: float*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:LEVel
driver.source.afRf.generator.voip.set_level(level = 1.0)
```

Specifies the audio output level for the VoIP path. For noise signals provided by an internal generator, the maximum allowed level is reduced by the factor 1/sqrt(2) .

param level
Range: 0.01 % to 100 %, Unit: %

set_value(*vo_ip_source: VoIpSource*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP
driver.source.afRf.generator.voip.set_value(vo_ip_source = enums.VoIpSource.
↪AFI1)
```

Selects an audio signal source for the VoIP path.

param vo_ip_source
GEN4 | AFI1 | AFI2 GEN4 Audio generator 4

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.afRf.generator.voip.clone()
```

Subgroups

6.15.1.1.22.1 AtmFrequency

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:VOIP:ATMFrequency:ENABle
```

class AtmFrequencyCls

AtmFrequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:ATMFrequency:ENABle
value: bool = driver.source.afRf.generator.voip.atmFrequency.get_enable()
```

Copies the current value of the carrier center frequency of AFRF generator to the RF measurements Also, copies the frequency value when changing the carrier frequency value via the FID.

return
atm_frequency: OFF | ON

set_enable(*atm_frequency: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:ATMFrequency:ENABle
driver.source.afRf.generator.voip.atmFrequency.set_enable(atm_frequency = False)
```

Copies the current value of the carrier center frequency of AFRF generator to the RF measurements Also, copies the frequency value when changing the carrier frequency value via the FID.

param atm_frequency
OFF | ON

6.15.1.1.22.2 Ptt

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:VOIP:PTT:STATe
SOURce:AFRF:GENerator<Instance>:VOIP:PTT
```

class PttCls

Ptt commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_state() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:PTT:STATe
value: bool = driver.source.afRf.generator.voip.ptt.get_state()
```

Sets the DUT's PTT state. Disable PTT at the DUT side, if you are finished with the TX testing.

return
ptt_state: OFF | ON

get_value() → bool

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:PTT
value: bool = driver.source.afRf.generator.voip.ptt.get_value()
```

Enables or disables the PTT state.

return
ptt: OFF | ON

set_value(*ptt: bool*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:PTT
driver.source.afRf.generator.voip.ptt.set_value(ptt = False)
```

Enables or disables the PTT state.

param ptt
OFF | ON

6.15.1.1.22.3 Sip

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RESPonse
SOURce:AFRF:GENerator<Instance>:VOIP:SIP:STATe
SOURce:AFRF:GENerator<Instance>:VOIP:SIP:CODE
SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RPRotocol
SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RCAuse
SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RTEXt
```

class SipCls

Sip commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_code() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:SIP:CODE
value: int = driver.source.afRf.generator.voip.sip.get_code()
```

Queries the code number of the last received SIP response.

return
sip_code: Decimal number, for example 200

get_rcause() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RCAuse
value: str = driver.source.afRf.generator.voip.sip.get_rcause()
```

Queries the reason (cause) of the VoIP connection.

return
rcause: No help available

get_response() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RESPonse
value: str = driver.source.afRf.generator.voip.sip.get_response()
```

Queries the text of the last received SIP response.

return
sip_response: Response string, for example 'OK'

get_rprotocol() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RPRotocol
value: str = driver.source.afRf.generator.voip.sip.get_rprotocol()
```

Queries the reason (protocol) of the VoIP connection.

return
protocol: No help available

get_rt_ext() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:SIP:RTEXt
value: str = driver.source.afRf.generator.voip.sip.get_rt_ext()
```

Queries the reason (text) of the VoIP connection.

return
rt_ext: No help available

get_state() → SipState

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:SIP:STATe
value: enums.SipState = driver.source.afRf.generator.voip.sip.get_state()
```

Queries the state of the VoIP connection to the DUT.

return
sip_state: TERMinated | ESTablished | ERRor

6.15.1.1.22.4 Uri

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:VOIP:URI:PORT
SOURce:AFRF:GENerator<Instance>:VOIP:URI:CMA
SOURce:AFRF:GENerator<Instance>:VOIP:URI:IP
SOURce:AFRF:GENerator<Instance>:VOIP:URI:USER
```

class UriCls

Uri commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_cma() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:CMA
value: str = driver.source.afRf.generator.voip.uri.get_cma()
```

Specifies the <user> part of the URI of the CMA (‘sip:<user>@<IP address>’) .

return
cma_uri: String with user part

get_ip() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:IP
value: str = driver.source.afRf.generator.voip.uri.get_ip()
```

Specifies the <IP address> part of the URI of the DUT (‘sip:<user>@<IP address>’) .

return
uri_ip: IP address as string

get_port() → int

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:PORT
value: int = driver.source.afRf.generator.voip.uri.get_port()
```

Specifies the URI port number of the DUT.

return
port: Range: 1024 to 65.535E+3

get_user() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:USER
value: str = driver.source.afRf.generator.voip.uri.get_user()
```

Specifies the <user> part of the URI of the DUT (‘sip:<user>@<IP address>’) .

return
uri_user: String with user part

set_cma(*cma_uri: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:CMA
driver.source.afRf.generator.voip.uri.set_cma(cma_uri = '1')
```

Specifies the <user> part of the URI of the CMA (‘sip:<user>@<IP address>’) .

param cma_uri
String with user part

set_ip(*uri_ip: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:IP
driver.source.afRf.generator.voip.uri.set_ip(uri_ip = '1')
```

Specifies the <IP address> part of the URI of the DUT (‘sip:<user>@<IP address>’) .

param uri_ip
IP address as string

set_port(*port: int*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:PORT
driver.source.afRf.generator.voip.uri.set_port(port = 1)
```

Specifies the URI port number of the DUT.

param port
Range: 1024 to 65.535E+3

set_user(*uri_user: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:VOIP:URI:USER
driver.source.afRf.generator.voip.uri.set_user(uri_user = '1')
```

Specifies the <user> part of the URI of the DUT (‘sip:<user>@<IP address>’) .

param uri_user
String with user part

6.15.1.1.23 Zigbee

SCPI Command:

```
SOURce:AFRF:GENerator<Instance>:ZIGBee:SNUMber
SOURce:AFRF:GENerator<Instance>:ZIGBee:DPAN
SOURce:AFRF:GENerator<Instance>:ZIGBee:DADDress
SOURce:AFRF:GENerator<Instance>:ZIGBee:SPAN
SOURce:AFRF:GENerator<Instance>:ZIGBee:SADDress
SOURce:AFRF:GENerator<Instance>:ZIGBee:PAYLoad
SOURce:AFRF:GENerator<Instance>:ZIGBee:MODE
SOURce:AFRF:GENerator<Instance>:ZIGBee:SDEViation
SOURce:AFRF:GENerator<Instance>:ZIGBee:SRATe
```

class ZigbeeCls

Zigbee commands group definition. 9 total commands, 0 Subgroups, 9 group commands

get_daddress() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:DADDress
value: str = driver.source.afRf.generator.zigbee.get_daddress()
```

Configures the destination address, i.e. the DUT’s address, to be signaled to the DUT, for ZigBee.

return
daddr: decimal Range: 0 to 65.535E+3

get_dpan() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:DPAN
value: str = driver.source.afRf.generator.zigbee.get_dpan()
```

Configures the destination ID of the private area network (PAN) signaled to the DUT, for ZIGBee.

return
dpan: Range: 0 to 65.535E+3

get_mode() → ZigBeeMode

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:MODE
value: enums.ZigBeeMode = driver.source.afRf.generator.zigbee.get_mode()
```

Queries the modulation type used for ZIGBee.

return
mode: OQPSk

get_payload() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:PAYLoad
value: str = driver.source.afRf.generator.zigbee.get_payload()
```

Configures the payload data to be signaled to the DUT, for ZIGBee.

return
payload: No help available

get_saddress() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SADDress
value: str = driver.source.afRf.generator.zigbee.get_saddress()
```

Configures the source address, i.e. the address of the CMA, signaled to the DUT, for ZIGBee.

return
saddress: Range: 0 to 65.535E+3

get_snumber() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SNUMber
value: str = driver.source.afRf.generator.zigbee.get_snumber()
```

Configures the sequence number, for ZIGBee.

return
snum: Range: 0 to 255

get_span() → str

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SPAN
value: str = driver.source.afRf.generator.zigbee.get_span()
```

Sets the source ID of the private area network (PAN) signaled to the DUT, for ZIGBee.

return
span: Range: 0 to 65.535E+3

get_standard_dev() → List[float]

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SDEViation
value: List[float] = driver.source.afRf.generator.zigbee.get_standard_dev()
```

Queries the frequency deviations used for OQPSK modulation, for ZIGBee.

return
sdeviation: Range: -180 deg to 180 deg, Unit: deg

get_symbol_rate() → float

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SRATe
value: float = driver.source.afRf.generator.zigbee.get_symbol_rate()
```

Queries the symbol rate resulting from the configured transmission mode, for ZIGBee.

return
srate: Range: 0 symbol/s to 1E+6 symbol/s, Unit: bit/s

set_daddress(*daddr: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:DADDress
driver.source.afRf.generator.zigbee.set_daddress(daddr = r1)
```

Configures the destination address, i.e. the DUT's address, to be signaled to the DUT, for ZigBee.

param daddr
decimal Range: 0 to 65.535E+3

set_dpan(*dpan: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:DPAN
driver.source.afRf.generator.zigbee.set_dpan(dpan = r1)
```

Configures the destination ID of the private area network (PAN) signaled to the DUT, for ZIGBee.

param dpan
Range: 0 to 65.535E+3

set_payload(*payload: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:PAYLoad
driver.source.afRf.generator.zigbee.set_payload(payload = '1')
```

Configures the payload data to be signaled to the DUT, for ZIGBee.

param payload
No help available

set_saddress(*saddress: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SADDress
driver.source.afRf.generator.zigbee.set_saddress(saddress = r1)
```

Configures the source address, i.e. the address of the CMA, signaled to the DUT, for ZIGBee.

param saddress
Range: 0 to 65.535E+3

set_snumber(*snum: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SNUMber
driver.source.afRf.generator.zigbee.set_snumber(snum = r1)
```

Configures the sequence number, for ZIGBee.

param snum
Range: 0 to 255

set_span(*span: str*) → None

```
# SCPI: SOURce:AFRF:GENerator<Instance>:ZIGBee:SPAN
driver.source.afRf.generator.zigbee.set_span(span = r1)
```

Sets the source ID of the private area network (PAN) signaled to the DUT, for ZIGBee.

param span
Range: 0 to 65.535E+3

6.15.2 Avionics

class AvionicsCls

Avionics commands group definition. 76 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.clone()
```

Subgroups

6.15.2.1 Generator

class GeneratorCls

Generator commands group definition. 76 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.clone()
```

Subgroups

6.15.2.1.1 Ils

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:FPAirment
SOURce:AVIonics:GENerator<Instance>:ILS
```

class IlsCls

Ils commands group definition. 39 total commands, 3 Subgroups, 2 group commands

get_fpairment() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:FPAirment
value: bool = driver.source.avionics.generator.ils.get_fpairment()
```

Enables or disables 'Frequency Pairment', that is the coupling between the glide slope carrier frequency and the localizer carrier frequency.

return
pairment: OFF | ON

get_value() → IlsTab

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS
value: enums.IlsTab = driver.source.avionics.generator.ils.get_value()
```

Selects the ILS generator subtab to be displayed at the GUI.

return
ils_tab: LOCalizer | GSLope

set_fpairment(*pairment: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:FPAirment
driver.source.avionics.generator.ils.set_fpairment(pairment = False)
```

Enables or disables 'Frequency Pairment', that is the coupling between the glide slope carrier frequency and the localizer carrier frequency.

param pairment
OFF | ON

set_value(*ils_tab: IlsTab*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS
driver.source.avionics.generator.ils.set_value(ils_tab = enums.IlsTab.GSLope)
```

Selects the ILS generator subtab to be displayed at the GUI.

param ils_tab
LOCalizer | GSLope

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.clone()
```

Subgroups

6.15.2.1.1.1 Gslope

class GslopeCls

Gslope commands group definition. 16 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.gslope.clone()
```

Subgroups

6.15.2.1.1.2 AfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:CONNector
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:ENABle
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:SDM
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:DDM
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:POFFset
```

class AfSettingsCls

AfSettings commands group definition. 12 total commands, 4 Subgroups, 5 group commands

get_connector() → AudioConnector

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:CONNector
value: enums.AudioConnector = driver.source.avionics.generator.ils.gslope.
↪afSettings.get_connector()
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) . If you want to route both the localizer signal and the glide slope signal to an AF output, you must configure different connectors for the two signals.

return
connector: AF1O | AF2O

get_ddm() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:DDM
value: float = driver.source.avionics.generator.ils.gslope.afSettings.get_ddm()
```

Sets the difference in modulation depth between the two lobes. The maximum allowed absolute value is limited by the configured SDM value.

return
ddm: Range: -100 % to 100 %, Unit: %

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:ENABle
value: bool = driver.source.avionics.generator.ils.gslope.afSettings.get_
↪enable()
```

Enables or disables the modulation of the RF carrier with the audio tones for the two lobes.

return
enable: OFF | ON

get_poffset() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:POFFset
value: float = driver.source.avionics.generator.ils.gslope.afSettings.get_
↪poffset()
```

Sets the phase offset between the audio signals of the two lobes.

return
poffset: Range: -60 deg to 120 deg, Unit: deg

get_sdm() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:SDM
value: float = driver.source.avionics.generator.ils.gslope.afSettings.get_sdm()
```

Sets the sum of depth of modulations (SDM) .

return
mod_depth: Range: 0 % to 100 %, Unit: %

set_connector(*connector: AudioConnector*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:CONNector
driver.source.avionics.generator.ils.gslope.afSettings.set_connector(connector␣
↪= enums.AudioConnector.AF1O)
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) . If you want to route both the localizer signal and the glide slope signal to an AF output, you must configure different connectors for the two signals.

param connector
AF1O | AF2O

set_ddm(*ddm: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:DDM
driver.source.avionics.generator.ils.gslope.afSettings.set_ddm(ddm = 1.0)
```

Sets the difference in modulation depth between the two lobes. The maximum allowed absolute value is limited by the configured SDM value.

param ddm
Range: -100 % to 100 %, Unit: %

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:ENABle
driver.source.avionics.generator.ils.gslope.afSettings.set_enable(enable =␣
↪False)
```

Enables or disables the modulation of the RF carrier with the audio tones for the two lobes.

param enable
OFF | ON

set_poffset(*poffset: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:POFFset
driver.source.avionics.generator.ils.gslope.afSettings.set_poffset(poffset = 1.
↪0)
```

Sets the phase offset between the audio signals of the two lobes.

param poffset
Range: -60 deg to 120 deg, Unit: deg

set_sdm(*mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:SDM
driver.source.avionics.generator.ils.gslope.afSettings.set_sdm(mod_depth = 1.0)
```

Sets the sum of depth of modulations (SDM) .

param mod_depth
Range: 0 % to 100 %, Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.gslope.afSettings.clone()
```

Subgroups

6.15.2.1.1.3 AudioOutput

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:AOUT:ENABle
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:AOUT:LEVel
```

class AudioOutputCls

AudioOutput commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:AOUT:ENABle
value: bool = driver.source.avionics.generator.ils.gslope.afSettings.
↪audioOutput.get_enable()
```

Enables or disables the AF output path for the glide slope signal.

return
af_enable: OFF | ON

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:AOUT:LEVel
value: float = driver.source.avionics.generator.ils.gslope.afSettings.
↪audioOutput.get_level()
```

Specifies the output level for the AF output path.

return
level: Range: 10E-6 V to 5 V, Unit: V

set_enable(*af_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:AOUT:ENABle
driver.source.avionics.generator.ils.gslope.afSettings.audioOutput.set_
↪enable(af_enable = False)
```

Enables or disables the AF output path for the glide slope signal.

param af_enable
OFF | ON

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:AOUT:LEVel
driver.source.avionics.generator.ils.gslope.afSettings.audioOutput.set_
↪level(level = 1.0)
```

Specifies the output level for the AF output path.

param level
Range: 10E-6 V to 5 V, Unit: V

6.15.2.1.1.4 Fly

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FLY:IDIRection
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FLY
```

class FlyCls

Fly commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_idirection() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FLY:IDIRection
value: bool = driver.source.avionics.generator.ils.gslope.afSettings.fly.get_
↪idirection()
```

Inverts the current direction towards the ideal line (fly up or fly down) .

return
invert_direction: OFF | ON

get_value() → UpDownDirection

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FLY
value: enums.UpDownDirection = driver.source.avionics.generator.ils.gslope.
↪afSettings.fly.get_value()
```

Sets the direction towards the ideal line (fly up or fly down) and the sign of the configured DDM value.

return
direction: UP | DOWN

set_idirection(*invert_direction: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FLY:IDIRection
driver.source.avionics.generator.ils.gslope.afSettings.fly.set_
↪idirection(invert_direction = False)
```

Inverts the current direction towards the ideal line (fly up or fly down) .

param invert_direction
OFF | ON

set_value(*direction: UpDownDirection*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FLY
driver.source.avionics.generator.ils.gslope.afSettings.fly.set_value(direction␣
↪= enums.UpDownDirection.DOWN)
```

Sets the direction towards the ideal line (fly up or fly down) and the sign of the configured DDM value.

param direction
UP | DOWN

6.15.2.1.1.5 Fmoddepth

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FMODdepth<nr>
```

class FmoddepthCls

Fmoddepth commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*frequencyLobe=FrequencyLobe.Nr1*) → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FMODdepth<nr>
value: float = driver.source.avionics.generator.ils.gslope.afSettings.fmoddepth.
↪get(frequencyLobe = repcap.FrequencyLobe.Nr1)
```

Queries the modulation depth for one lobe.

param frequencyLobe
optional repeated capability selector. Default value: Nr1

return
freq_1: Range: 0 % to 100 %, Unit: %

6.15.2.1.1.6 Frequency<FrequencyLobe>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.source.avionics.generator.ils.gslope.afSettings.frequency.repcap_
↪frequencyLobe_get()
driver.source.avionics.generator.ils.gslope.afSettings.frequency.repcap_frequencyLobe_
↪set(repcap.FrequencyLobe.Nr1)
```

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FREQuency<nr>
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: FrequencyLobe, default value after init: FrequencyLobe.Nr1

get(*frequencyLobe=FrequencyLobe.Default*) → int

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FREQuency<nr>
value: int = driver.source.avionics.generator.ils.gslope.afSettings.frequency.
↪get(frequencyLobe = repcap.FrequencyLobe.Default)
```

Configures the audio frequency for one lobe.

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
freq_1: Range and reset value depend on no Range: 72 Hz to 108 Hz / 120 Hz to 180 Hz , Unit: Hz

set(*freq_1: int*, *frequencyLobe=FrequencyLobe.Default*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FREQuency<nr>
driver.source.avionics.generator.ils.gslope.afSettings.frequency.set(freq_1 = 1,
↪ frequencyLobe = repcap.FrequencyLobe.Default)
```

Configures the audio frequency for one lobe.

param freq_1
Range and reset value depend on no Range: 72 Hz to 108 Hz / 120 Hz to 180 Hz , Unit: Hz

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.gslope.afSettings.frequency.clone()
```

Subgroups

6.15.2.1.1.7 Enable

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FREQuency<nr>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*frequencyLobe=FrequencyLobe.Default*) → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FREQuency<nr>
↪:ENABle
value: bool = driver.source.avionics.generator.ils.gslope.afSettings.frequency.
↪enable.get(frequencyLobe = repcap.FrequencyLobe.Default)
```

Enables or disables lobe number <no>. At least one lobe must be enabled.

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
freq_1: OFF | ON

set(*freq_1: bool*, *frequencyLobe=FrequencyLobe.Default*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:AFSettings:FREQuency<nr>
↪:ENABle
driver.source.avionics.generator.ils.gslope.afSettings.frequency.enable.
↪set(freq_1 = False, frequencyLobe = repcap.FrequencyLobe.Default)
```

Enables or disables lobe number <no>. At least one lobe must be enabled.

param freq_1
OFF | ON

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

6.15.2.1.1.8 RfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:FREQuency
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:LEVel
```

class RfSettingsCls

RfSettings commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:FREQuency
value: float = driver.source.avionics.generator.ils.gslope.rfSettings.get_
↪frequency()
```

Specifies the center frequency of the unmodulated RF carrier for the glide slope signal.

return
freq: Range: 100 kHz to 3 GHz, Unit: Hz

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:LEVel
value: float = driver.source.avionics.generator.ils.gslope.rfSettings.get_
↪level()
```

Specifies the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

return
level: Unit: dBm

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:FREQuency
driver.source.avionics.generator.ils.gslope.rfSettings.set_frequency(freq = 1.0)
```

Specifies the center frequency of the unmodulated RF carrier for the glide slope signal.

param freq
Range: 100 kHz to 3 GHz, Unit: Hz

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:LEVel
driver.source.avionics.generator.ils.gslope.rfSettings.set_level(level = 1.0)
```

Specifies the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

param level
Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.gslope.rfSettings.clone()
```

Subgroups

6.15.2.1.1.9 Channel

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:CHANnel
```

class ChannelCls

Channel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ChannelStruct

Response structure. Fields:

- Channel: int: Channel number Range: 18 to 56
- Letter: enums.IlsLetter: X | Y Channel letter

get() → ChannelStruct

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:CHANnel
value: ChannelStruct = driver.source.avionics.generator.ils.gslope.rfSettings.
↪channel.get()
```

Selects the RF channel. Each channel is identified via a number and a letter, for example 18X.

return
structure: for return value, see the help for ChannelStruct structure arguments.

set(*channel: int*, *letter: IlsLetter*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:CHANnel
driver.source.avionics.generator.ils.gslope.rfSettings.channel.set(channel = 1,␣
↪letter = enums.IlsLetter.X)
```

Selects the RF channel. Each channel is identified via a number and a letter, for example 18X.

param channel
Channel number Range: 18 to 56

param letter
X | Y Channel letter

6.15.2.1.1.10 RfOut

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:RFOut:ENABle
```

class RfOutCls

RfOut commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:RFOut:ENABle
value: bool = driver.source.avionics.generator.ils.gslope.rfSettings.rfOut.get_
↪enable()
```

Enables or disables the RF output path for the glide slope signal.

return
rf_enable: OFF | ON

set_enable(*rf_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:GSLope:RFSettings:RFOut:ENABle
driver.source.avionics.generator.ils.gslope.rfSettings.rfOut.set_enable(rf_
↪enable = False)
```

Enables or disables the RF output path for the glide slope signal.

param rf_enable
OFF | ON

6.15.2.1.1.11 Localizer

class LocalizerCls

Localizer commands group definition. 19 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.localizer.clone()
```

Subgroups

6.15.2.1.1.12 AfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:CONNector
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:ENABle
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:SDM
```

(continues on next page)

(continued from previous page)

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:DDM
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:POFFset
```

class AfSettingsCls

AfSettings commands group definition. 12 total commands, 4 Subgroups, 5 group commands

get_connector() → AudioConnector

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:CONNector
value: enums.AudioConnector = driver.source.avionics.generator.ils.localizer.
↪afSettings.get_connector()
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) . If you want to route both the localizer signal and the glide slope signal to an AF output, you must configure different connectors for the two signals.

return
connector: AF1O | AF2O

get_ddm() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:DDM
value: float = driver.source.avionics.generator.ils.localizer.afSettings.get_
↪ddm()
```

Sets the difference in modulation depth between the two lobes. The maximum allowed absolute value is limited by the configured SDM value.

return
ddm: Range: -100 % to 100 %, Unit: %

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:ENABle
value: bool = driver.source.avionics.generator.ils.localizer.afSettings.get_
↪enable()
```

Enables or disables the modulation of the RF carrier with the audio tones for the two lobes.

return
enable: OFF | ON

get_poffset() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:POFFset
value: float = driver.source.avionics.generator.ils.localizer.afSettings.get_
↪poffset()
```

Sets the phase offset between the audio signals of the two lobes.

return
poffset: Range: -60 deg to 120 deg, Unit: deg

get_sdm() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:SDM
value: float = driver.source.avionics.generator.ils.localizer.afSettings.get_
↪sdm()
```

Sets the sum of depth of modulations (SDM) .

return
mod_depth: Range: 0 % to 100 %, Unit: %

set_connector(*connector: AudioConnector*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:CONNector
driver.source.avionics.generator.ils.localizer.afSettings.set_
↪connector(connector = enums.AudioConnector.AF10)
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) . If you want to route both the localizer signal and the glide slope signal to an AF output, you must configure different connectors for the two signals.

param connector
AF1O | AF2O

set_ddm(*ddm: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:DDM
driver.source.avionics.generator.ils.localizer.afSettings.set_ddm(ddm = 1.0)
```

Sets the difference in modulation depth between the two lobes. The maximum allowed absolute value is limited by the configured SDM value.

param ddm
Range: -100 % to 100 %, Unit: %

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:ENABle
driver.source.avionics.generator.ils.localizer.afSettings.set_enable(enable =␣
↪False)
```

Enables or disables the modulation of the RF carrier with the audio tones for the two lobes.

param enable
OFF | ON

set_poffset(*poffset: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:POFFset
driver.source.avionics.generator.ils.localizer.afSettings.set_poffset(poffset =␣
↪1.0)
```

Sets the phase offset between the audio signals of the two lobes.

param poffset
Range: -60 deg to 120 deg, Unit: deg

set_sdm(*mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:SDM
driver.source.avionics.generator.ils.localizer.afSettings.set_sdm(mod_depth = 1.
↪0)
```

Sets the sum of depth of modulations (SDM) .

param mod_depth
Range: 0 % to 100 %, Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.localizer.afSettings.clone()
```

Subgroups

6.15.2.1.1.13 AudioOutput

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:AOUT:ENABle
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:AOUT:LEVel
```

class AudioOutputCls

AudioOutput commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:AOUT:ENABle
value: bool = driver.source.avionics.generator.ils.localizer.afSettings.
↪audioOutput.get_enable()
```

Enables or disables the AF output path for the localizer signal.

return
af_enable: OFF | ON

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:AOUT:LEVel
value: float = driver.source.avionics.generator.ils.localizer.afSettings.
↪audioOutput.get_level()
```

Specifies the output level for the AF output path.

return
level: Range: 10E-6 V to 5 V, Unit: V

set_enable(*af_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:AOUT:ENABle
driver.source.avionics.generator.ils.localizer.afSettings.audioOutput.set_
↪enable(af_enable = False)
```

Enables or disables the AF output path for the localizer signal.

param af_enable
OFF | ON

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:AOUT:LEVel
driver.source.avionics.generator.ils.localizer.afSettings.audioOutput.set_
↪level(level = 1.0)
```

Specifies the output level for the AF output path.

param level
Range: 10E-6 V to 5 V, Unit: V

6.15.2.1.1.14 Fly

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FLY:IDIRection
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FLY
```

class FlyCls

Fly commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_idirection() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>
↪:ILS:LOCalizer:AFSettings:FLY:IDIRection
value: bool = driver.source.avionics.generator.ils.localizer.afSettings.fly.get_
↪idirection()
```

Inverts the current direction towards the ideal line (fly left or fly right) .

return
invert_direction: OFF | ON

get_value() → LeftRightDirection

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FLY
value: enums.LeftRightDirection = driver.source.avionics.generator.ils.
↪localizer.afSettings.fly.get_value()
```

Sets the direction towards the ideal line (fly left or fly right) and the sign of the configured DDM value.

return
direction: LEFT | RIGHt

set_idirection(*invert_direction: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>
↪:ILS:LOCalizer:AFSettings:FLY:IDIRection
driver.source.avionics.generator.ils.localizer.afSettings.fly.set_
↪idirection(invert_direction = False)
```

Inverts the current direction towards the ideal line (fly left or fly right) .

param invert_direction
OFF | ON

`set_value`(*direction: LeftRightDirection*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FLY
driver.source.avionics.generator.ils.localizer.afSettings.fly.set_
↪value(direction = enums.LeftRightDirection.LEFT)
```

Sets the direction towards the ideal line (fly left or fly right) and the sign of the configured DDM value.

param direction
LEFT | RIGHt

6.15.2.1.1.15 Fmoddepth

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FMODdepth<nr>
```

`class FmoddepthCls`

Fmoddepth commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`get`(*frequencyLobe=FrequencyLobe.Nr1*) → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FMODdepth
↪<nr>
value: float = driver.source.avionics.generator.ils.localizer.afSettings.
↪fmoddepth.get(frequencyLobe = repcap.FrequencyLobe.Nr1)
```

Queries the modulation depth for one lobe.

param frequencyLobe
optional repeated capability selector. Default value: Nr1

return
freq_1: Range: 0 % to 100 %, Unit: %

6.15.2.1.1.16 Frequency<FrequencyLobe>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.source.avionics.generator.ils.localizer.afSettings.frequency.repcap_
↪frequencyLobe_get()
driver.source.avionics.generator.ils.localizer.afSettings.frequency.repcap_frequencyLobe_
↪set(repcap.FrequencyLobe.Nr1)
```

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FREQuency<nr>
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: FrequencyLobe, default value after init: FrequencyLobe.Nr1

get(*frequencyLobe=FrequencyLobe.Default*) → int

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FREQuency
↪<nr>
value: int = driver.source.avionics.generator.ils.localizer.afSettings.
↪frequency.get(frequencyLobe = repcap.FrequencyLobe.Default)
```

Configures the audio frequency for one lobe.

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
freq_1: Range and reset value depend on no Range: 72 Hz to 108 Hz / 120 Hz to 180 Hz , Unit: Hz

set(*freq_1: int*, *frequencyLobe=FrequencyLobe.Default*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FREQuency
↪<nr>
driver.source.avionics.generator.ils.localizer.afSettings.frequency.set(freq_1␣
↪= 1, frequencyLobe = repcap.FrequencyLobe.Default)
```

Configures the audio frequency for one lobe.

param freq_1
Range and reset value depend on no Range: 72 Hz to 108 Hz / 120 Hz to 180 Hz , Unit: Hz

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.localizer.afSettings.frequency.clone()
```

Subgroups

6.15.2.1.1.17 Enable

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FREQuency<nr>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*frequencyLobe=FrequencyLobe.Default*) → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FREQuency
↪<nr>:ENABle
value: bool = driver.source.avionics.generator.ils.localizer.afSettings.
↪frequency.enable.get(frequencyLobe = repcap.FrequencyLobe.Default)
```

Enables or disables lobe number <no>. At least one lobe must be enabled.

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

return
freq_1: OFF | ON

set(*freq_1: bool*, *frequencyLobe=FrequencyLobe.Default*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:AFSettings:FREQuency
↪<nr>:ENABle
driver.source.avionics.generator.ils.localizer.afSettings.frequency.enable.
↪set(freq_1 = False, frequencyLobe = repcap.FrequencyLobe.Default)
```

Enables or disables lobe number <no>. At least one lobe must be enabled.

param freq_1
OFF | ON

param frequencyLobe
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Frequency')

6.15.2.1.1.18 IdSignal

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:ENABle
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:MDEPth
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:FREQuency
```

class IdSignalCls

IdSignal commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:ENABle
value: bool = driver.source.avionics.generator.ils.localizer.idSignal.get_
↪enable()
```

Enables or disables the ID signal.

return
enable: OFF | ON

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:FREQuency
value: float = driver.source.avionics.generator.ils.localizer.idSignal.get_
↪frequency()
```

Configures the audio frequency of the ID signal.

return
freq: Range: 0 Hz to 21 kHz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:MDEPth
value: float = driver.source.avionics.generator.ils.localizer.idSignal.get_mod_
↪depth()
```

Configures the modulation depth for the ID signal. The sum of the SDM and of the modulation depth for the ID signal must not exceed 100 % (if both signals are enabled) .

return
mod_depth: Range: 0 % to 100 %, Unit: %

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:ENABle
driver.source.avionics.generator.ils.localizer.idSignal.set_enable(enable =␣
↪False)
```

Enables or disables the ID signal.

param enable
OFF | ON

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:FREQuency
driver.source.avionics.generator.ils.localizer.idSignal.set_frequency(freq = 1.
↪0)
```

Configures the audio frequency of the ID signal.

param freq
Range: 0 Hz to 21 kHz, Unit: Hz

set_mod_depth(*mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:IDSignal:MDEPth
driver.source.avionics.generator.ils.localizer.idSignal.set_mod_depth(mod_depth␣
↪= 1.0)
```

Configures the modulation depth for the ID signal. The sum of the SDM and of the modulation depth for the ID signal must not exceed 100 % (if both signals are enabled) .

param mod_depth
Range: 0 % to 100 %, Unit: %

6.15.2.1.1.19 RfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:FREQuency
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:LEVel
```

class RfSettingsCls

RfSettings commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:FREQuency
value: float = driver.source.avionics.generator.ils.localizer.rfSettings.get_
↪frequency()
```

Specifies the center frequency of the unmodulated RF carrier for the localizer signal.

return
freq: Range: 100 kHz to 3 GHz, Unit: Hz

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:LEVel
value: float = driver.source.avionics.generator.ils.localizer.rfSettings.get_
↪level()
```

Specifies the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

return
level: Unit: dBm

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:FREQuency
driver.source.avionics.generator.ils.localizer.rfSettings.set_frequency(freq =␣
↪1.0)
```

Specifies the center frequency of the unmodulated RF carrier for the localizer signal.

param freq
Range: 100 kHz to 3 GHz, Unit: Hz

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:LEVel
driver.source.avionics.generator.ils.localizer.rfSettings.set_level(level = 1.0)
```

Specifies the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

param level
Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.ils.localizer.rfSettings.clone()
```

Subgroups

6.15.2.1.1.20 Channel

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:CHANnel
```

class ChannelCls

Channel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class ChannelStruct

Response structure. Fields:

- Channel: int: Channel number Range: 18 to 56
- Letter: enums.IlsLetter: X | Y Channel letter

get() → ChannelStruct

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:CHANnel
value: ChannelStruct = driver.source.avionics.generator.ils.localizer.
↪rfSettings.channel.get()
```

Selects the RF channel. Each channel is identified via a number and a letter, for example 18X.

return
structure: for return value, see the help for ChannelStruct structure arguments.

set(*channel: int*, *letter: IlsLetter*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:CHANnel
driver.source.avionics.generator.ils.localizer.rfSettings.channel.set(channel =␣
↪1, letter = enums.IlsLetter.X)
```

Selects the RF channel. Each channel is identified via a number and a letter, for example 18X.

param channel
Channel number Range: 18 to 56

param letter
X | Y Channel letter

6.15.2.1.1.21 RfOut

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:LOCalizer:RFSettings:RFOut:ENABle
```

class RfOutCls

RfOut commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>
↪:ILS:LOCalizer:RFSettings:RFOut:ENABle
value: bool = driver.source.avionics.generator.ils.localizer.rfSettings.rfOut.
↪get_enable()
```

Enables or disables the RF output path for the localizer signal.

return
rf_enable: OFF | ON

set_enable(*rf_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>
↪:ILS:LOCalizer:RFSettings:RFOut:ENABle
driver.source.avionics.generator.ils.localizer.rfSettings.rfOut.set_enable(rf_
↪enable = False)
```

Enables or disables the RF output path for the localizer signal.

param rf_enable
OFF | ON

6.15.2.1.1.22 State

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:ILS:STATe
SOURce:AVIonics:GENerator<Instance>:ILS:STATe:ALL
```

class StateCls

State commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get() → GeneratorState

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:STATe
value: enums.GeneratorState = driver.source.avionics.generator.ils.state.get()
```

Starts or stops the ILS generator or queries its state.

return
gen_state: OFF | ON | PENDing OFF Generator is off ON Generator is running PENDing Start or stop of generator is ongoing

get_all() → List[GeneratorState]

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:STATe:ALL
value: List[enums.GeneratorState] = driver.source.avionics.generator.ils.state.
↪get_all()
```

No command help available

return
all_states: No help available

set(*gen_control: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:ILS:STATe
driver.source.avionics.generator.ils.state.set(gen_control = False)
```

Starts or stops the ILS generator or queries its state.

param gen_control
ON | OFF ON Starts the generator OFF Stops the generator

6.15.2.1.2 MarkerBeacon

class MarkerBeaconCls

MarkerBeacon commands group definition. 14 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.markerBeacon.clone()
```

Subgroups

6.15.2.1.2.1 AfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:CONNector
SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:ENABle
SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:MDEPth
SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:FREQuency
```

class AfSettingsCls

AfSettings commands group definition. 6 total commands, 1 Subgroups, 4 group commands

get_connector() → AudioConnector

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:CONNector
value: enums.AudioConnector = driver.source.avionics.generator.markerBeacon.
↪afSettings.get_connector()
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) .

return
connector: AF1O | AF2O

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:ENABle
value: bool = driver.source.avionics.generator.markerBeacon.afSettings.get_
↪enable()
```

Enables or disables the modulation of the RF carrier with the marker beacon audio tone.

return
enable: OFF | ON

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:FREQuency
value: float = driver.source.avionics.generator.markerBeacon.afSettings.get_
↪frequency()
```

Sets the audio frequency of the tone to be modulated to the carrier.

return
freq: Range: 0 Hz to 10 kHz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:MDEPth
value: float = driver.source.avionics.generator.markerBeacon.afSettings.get_mod_
↪depth()
```

Sets the modulation depth for amplitude modulation of the carrier. The sum of the modulation depths for all enabled components must not exceed 100 %.

return
mod_depth: Range: 0 % to 100 %, Unit: %

set_connector(*connector: AudioConnector*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:CONNector
driver.source.avionics.generator.markerBeacon.afSettings.set_
↪connector(connector = enums.AudioConnector.AF10)
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) .

param connector
AF1O | AF2O

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:ENABle
driver.source.avionics.generator.markerBeacon.afSettings.set_enable(enable =␣
↪False)
```

Enables or disables the modulation of the RF carrier with the marker beacon audio tone.

param enable
OFF | ON

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:FREQuency
driver.source.avionics.generator.markerBeacon.afSettings.set_frequency(freq = 1.
↪0)
```

Sets the audio frequency of the tone to be modulated to the carrier.

param freq
Range: 0 Hz to 10 kHz, Unit: Hz

set_mod_depth(*mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:MDEPth
driver.source.avionics.generator.markerBeacon.afSettings.set_mod_depth(mod_
↪depth = 1.0)
```

Sets the modulation depth for amplitude modulation of the carrier. The sum of the modulation depths for all enabled components must not exceed 100 %.

param mod_depth
Range: 0 % to 100 %, Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.markerBeacon.afSettings.clone()
```

Subgroups

6.15.2.1.2.2 AudioOutput

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:AOUT:ENABle
SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:AOUT:LEVel
```

class AudioOutputCls

AudioOutput commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:AOUT:ENABle
value: bool = driver.source.avionics.generator.markerBeacon.afSettings.
↪audioOutput.get_enable()
```

Enables or disables the AF output path.

return
af_enable: OFF | ON

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:AOUT:LEVel
value: float = driver.source.avionics.generator.markerBeacon.afSettings.
↪audioOutput.get_level()
```

Specifies the output level for the AF output path.

return
level: Range: 10E-6 V to 5 V, Unit: V

set_enable(*af_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:AOUT:ENABle
driver.source.avionics.generator.markerBeacon.afSettings.audioOutput.set_
↪enable(af_enable = False)
```

Enables or disables the AF output path.

param af_enable
OFF | ON

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:AFSettings:AOUT:LEVel
driver.source.avionics.generator.markerBeacon.afSettings.audioOutput.set_
↪level(level = 1.0)
```

Specifies the output level for the AF output path.

param level
Range: 10E-6 V to 5 V, Unit: V

6.15.2.1.2.3 IdSignal

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:ENABle
SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:MDEPth
SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:FREQuency
```

class IdSignalCls

IdSignal commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:ENABle
value: bool = driver.source.avionics.generator.markerBeacon.idSignal.get_
↪enable()
```

Enables or disables the ID signal.

return
enable: OFF | ON

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:FREQuency
value: float = driver.source.avionics.generator.markerBeacon.idSignal.get_
↪frequency()
```

Configures the frequency of the ID signal.

return
freq: Range: 0 Hz to 21 kHz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:MDEPth
value: float = driver.source.avionics.generator.markerBeacon.idSignal.get_mod_
↪depth()
```

Configures the modulation depth for the ID signal. The sum of the modulation depths for all enabled components must not exceed 100 %.

return
mod_depth: Range: 0 % to 100 %, Unit: %

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:ENABle
driver.source.avionics.generator.markerBeacon.idSignal.set_enable(enable =␣
↪False)
```

Enables or disables the ID signal.

param enable
OFF | ON

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:FREQuency
driver.source.avionics.generator.markerBeacon.idSignal.set_frequency(freq = 1.0)
```

Configures the frequency of the ID signal.

param freq
Range: 0 Hz to 21 kHz, Unit: Hz

set_mod_depth(*mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:IDSignal:MDEPth
driver.source.avionics.generator.markerBeacon.idSignal.set_mod_depth(mod_depth
↪= 1.0)
```

Configures the modulation depth for the ID signal. The sum of the modulation depths for all enabled components must not exceed 100 %.

param mod_depth
Range: 0 % to 100 %, Unit: %

6.15.2.1.2.4 RfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:FREQuency
SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:LEVel
```

class RfSettingsCls

RfSettings commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:FREQuency
value: float = driver.source.avionics.generator.markerBeacon.rfSettings.get_
↪frequency()
```

Sets the center frequency of the unmodulated RF carrier.

return
freq: Range: 100 kHz to 3 GHz, Unit: Hz

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:LEVel
value: float = driver.source.avionics.generator.markerBeacon.rfSettings.get_
↪level()
```

Sets the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

return
level: Unit: dBm

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:FREQuency
driver.source.avionics.generator.markerBeacon.rfSettings.set_frequency(freq = 1.
↪0)
```

Sets the center frequency of the unmodulated RF carrier.

param freq
Range: 100 kHz to 3 GHz, Unit: Hz

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:LEVel
driver.source.avionics.generator.markerBeacon.rfSettings.set_level(level = 1.0)
```

Sets the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

param level
Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.markerBeacon.rfSettings.clone()
```

Subgroups

6.15.2.1.2.5 RfOut

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:RFOut:ENABle
```

class RfOutCls

RfOut commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:RFOut:ENABle
value: bool = driver.source.avionics.generator.markerBeacon.rfSettings.rfOut.
↪get_enable()
```

Enables or disables the RF output path.

return
rf_enable: OFF | ON

set_enable(*rf_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:RFSettings:RFOut:ENABle
driver.source.avionics.generator.markerBeacon.rfSettings.rfOut.set_enable(rf_
↪enable = False)
```

Enables or disables the RF output path.

param rf_enable
OFF | ON

6.15.2.1.2.6 State

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:MBEacon:STATe
SOURce:AVIonics:GENerator<Instance>:MBEacon:STATe:ALL
```

class StateCls

State commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get() → GeneratorState

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:STATe
value: enums.GeneratorState = driver.source.avionics.generator.markerBeacon.
↪state.get()
```

Starts or stops the VOR generator or queries its state.

return
gen_state: OFF | ON | PENDing OFF Generator is off ON Generator is running PENDing Start or stop of generator is ongoing

get_all() → List[GeneratorState]

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:STATe:ALL
value: List[enums.GeneratorState] = driver.source.avionics.generator.
↪markerBeacon.state.get_all()
```

No command help available

return
all_states: No help available

set(*gen_control: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:MBEacon:STATe
driver.source.avionics.generator.markerBeacon.state.set(gen_control = False)
```

Starts or stops the VOR generator or queries its state.

param gen_control
ON | OFF ON Starts the generator OFF Stops the generator

6.15.2.1.3 RfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:RFSettings:CONNector
SOURce:AVIonics:GENerator<Instance>:RFSettings:EATTenuation
```

class RfSettingsCls

RfSettings commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_connector() → OutputConnector

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:RFSettings:CONNector
value: enums.OutputConnector = driver.source.avionics.generator.rfSettings.get_
↪connector()
```

Selects the output connector for the generated RF signal.

return
connector: RFCom | RFOut

get_eattenuation() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:RFSettings:EATTenuation
value: float = driver.source.avionics.generator.rfSettings.get_eattenuation()
```

Specifies the external attenuation in the RF output path. Negative values specify a gain.

return
ext_atten: Range: -50 dB to 90 dB, Unit: dB

set_connector(*connector: OutputConnector*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:RFSettings:CONNector
driver.source.avionics.generator.rfSettings.set_connector(connector = enums.
↪OutputConnector.RFCom)
```

Selects the output connector for the generated RF signal.

param connector
RFCom | RFOut

set_eattenuation(*ext_atten: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:RFSettings:EATTenuation
driver.source.avionics.generator.rfSettings.set_eattenuation(ext_atten = 1.0)
```

Specifies the external attenuation in the RF output path. Negative values specify a gain.

param ext_atten
Range: -50 dB to 90 dB, Unit: dB

6.15.2.1.4 Vor

class VorCls

Vor commands group definition. 21 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.vor.clone()
```

Subgroups

6.15.2.1.4.1 AfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:CONNector
```

class AfSettingsCls

AfSettings commands group definition. 13 total commands, 3 Subgroups, 1 group commands

get_connector() → AudioConnector

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:CONNector
value: enums.AudioConnector = driver.source.avionics.generator.vor.afSettings.
↪get_connector()
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) .

return
connector: AF1O | AF2O

set_connector(*connector: AudioConnector*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:CONNector
driver.source.avionics.generator.vor.afSettings.set_connector(connector = enums.
↪AudioConnector.AF1O)
```

Selects the output connector for the generated AF signal (AF1 OUT or AF2 OUT) .

param connector
AF1O | AF2O

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.vor.afSettings.clone()
```

Subgroups

6.15.2.1.4.2 AudioOutput

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:AOUT:ENABle
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:AOUT:LEVel
```

class AudioOutputCls

AudioOutput commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:AOUT:ENABle
value: bool = driver.source.avionics.generator.vor.afSettings.audioOutput.get_
↪enable()
```

Enables or disables the AF output path.

return
af_enable: OFF | ON

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:AOUT:LEVel
value: float = driver.source.avionics.generator.vor.afSettings.audioOutput.get_
↪level()
```

Specifies the output level for the AF output path.

return
level: Range: 10E-6 V to 5 V, Unit: V

set_enable(*af_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:AOUT:ENABle
driver.source.avionics.generator.vor.afSettings.audioOutput.set_enable(af_
↪enable = False)
```

Enables or disables the AF output path.

param af_enable
OFF | ON

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:AOUT:LEVel
driver.source.avionics.generator.vor.afSettings.audioOutput.set_level(level = 1.
↪0)
```

Specifies the output level for the AF output path.

param level
Range: 10E-6 V to 5 V, Unit: V

6.15.2.1.4.3 Reference

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:ENABle
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:MDEPth
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:CFRequency
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:FREQuency
```

class ReferenceCls

Reference commands group definition. 6 total commands, 1 Subgroups, 4 group commands

get_cfrequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:CFRequency
value: float = driver.source.avionics.generator.vor.afSettings.reference.get_
↪cfrequency()
```

Sets the center frequency of the FM subcarrier.

return
freq: Range: 7500 Hz to 12.5 kHz, Unit: Hz

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:ENABle
value: bool = driver.source.avionics.generator.vor.afSettings.reference.get_
↪enable()
```

Enables or disables the REF signal.

return
enable: OFF | ON

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:FREQuency
value: float = driver.source.avionics.generator.vor.afSettings.reference.get_
↪frequency()
```

Sets the audio frequency of the REF signal and the VAR signal.

return
freq: Range: 20 Hz to 40 Hz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:MDEPth
value: float = driver.source.avionics.generator.vor.afSettings.reference.get_
↪mod_depth()
```

Sets the AM modulation depth of the FM subcarrier. The sum of the modulation depths for all enabled components must not exceed 100 %.

return
vor_mod_depth: Range: 0 % to 100 %, Unit: %

set_cfrequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:CFRequency
driver.source.avionics.generator.vor.afSettings.reference.set_cfrequency(freq =␣
↪1.0)
```

Sets the center frequency of the FM subcarrier.

param freq
Range: 7500 Hz to 12.5 kHz, Unit: Hz

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:ENABle
driver.source.avionics.generator.vor.afSettings.reference.set_enable(enable =␣
↪False)
```

Enables or disables the REF signal.

param enable
OFF | ON

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:FREQuency
driver.source.avionics.generator.vor.afSettings.reference.set_frequency(freq =␣
↪1.0)
```

Sets the audio frequency of the REF signal and the VAR signal.

param freq
Range: 20 Hz to 40 Hz, Unit: Hz

set_mod_depth(*vor_mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:MDEPth
driver.source.avionics.generator.vor.afSettings.reference.set_mod_depth(vor_mod_
↪depth = 1.0)
```

Sets the AM modulation depth of the FM subcarrier. The sum of the modulation depths for all enabled components must not exceed 100 %.

param vor_mod_depth
Range: 0 % to 100 %, Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.vor.afSettings.reference.clone()
```

Subgroups

6.15.2.1.4.4 Fdeviation

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:FDEViation:ENABle
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:FDEViation
```

class FdeviationCls

Fdeviation commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>
↪:VOR:AFSettings:REFerence:FDEViation:ENABle
value: bool = driver.source.avionics.generator.vor.afSettings.reference.
↪fdeviation.get_enable()
```

Enables or disables the modulation of the FM subcarrier with the REF signal.

return
vor_freq_deviation: OFF | ON

get_value() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:FDEViation
value: float = driver.source.avionics.generator.vor.afSettings.reference.
↪fdeviation.get_value()
```

Sets the frequency deviation of the REF signal on the FM subcarrier.

return
vor_freq_deviation: Range: 300 Hz to 600 Hz, Unit: Hz

set_enable(*vor_freq_deviation: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>
↪:VOR:AFSettings:REFerence:FDEViation:ENABle
driver.source.avionics.generator.vor.afSettings.reference.fdeviation.set_
↪enable(vor_freq_deviation = False)
```

Enables or disables the modulation of the FM subcarrier with the REF signal.

param vor_freq_deviation
OFF | ON

set_value(*vor_freq_deviation: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:REFerence:FDEViation
driver.source.avionics.generator.vor.afSettings.reference.fdeviation.set_
↪value(vor_freq_deviation = 1.0)
```

Sets the frequency deviation of the REF signal on the FM subcarrier.

param vor_freq_deviation
Range: 300 Hz to 600 Hz, Unit: Hz

6.15.2.1.4.5 Vphase

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:ENABle
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:MDEPth
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:BANGle
SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:DIRection
```

class VphaseCls

Vphase commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_bangle() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:BANGle
value: float = driver.source.avionics.generator.vor.afSettings.vphase.get_
↪bangle()
```

Sets the bearing angle from the beacon to the receiver, or vice versa, depending on the configured direction.

return
vor_phase: Range: 0 deg to 360 deg, Unit: deg

get_direction() → VphaseDirection

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:DIRection
value: enums.VphaseDirection = driver.source.avionics.generator.vor.afSettings.
↪vphase.get_direction()
```

Sets the bearing direction.

return
direction: TO | FROM TO The bearing angle indicates the clockwise angle between north and the line from the receiver to the beacon. FROM The bearing angle indicates the clockwise angle between north and the line from the beacon to the receiver.

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:ENABle
value: bool = driver.source.avionics.generator.vor.afSettings.vphase.get_
↪enable()
```

Enables or disables the VAR signal.

return
enable: OFF | ON

get_mod_depth() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:MDEPth
value: float = driver.source.avionics.generator.vor.afSettings.vphase.get_mod_
↪depth()
```

Sets the AM modulation depth for the VAR signal. The sum of the modulation depths for all enabled components must not exceed 100 %.

return
vor_mod_depth: Range: 0 % to 100 %, Unit: %

set_bangle(*vor_phase: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:BANGle
driver.source.avionics.generator.vor.afSettings.vphase.set_bangle(vor_phase = 1.
↪0)
```

Sets the bearing angle from the beacon to the receiver, or vice versa, depending on the configured direction.

param vor_phase
Range: 0 deg to 360 deg, Unit: deg

set_direction(*direction: VphaseDirection*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:DIRection
driver.source.avionics.generator.vor.afSettings.vphase.set_direction(direction␣
↪= enums.VphaseDirection.FROM)
```

Sets the bearing direction.

param direction
TO | FROM TO The bearing angle indicates the clockwise angle between north and the line from the receiver to the beacon. FROM The bearing angle indicates the clockwise angle between north and the line from the beacon to the receiver.

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:ENABle
driver.source.avionics.generator.vor.afSettings.vphase.set_enable(enable =␣
↪False)
```

Enables or disables the VAR signal.

param enable
OFF | ON

set_mod_depth(*vor_mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:AFSettings:VPHase:MDEPth
driver.source.avionics.generator.vor.afSettings.vphase.set_mod_depth(vor_mod_
↪depth = 1.0)
```

Sets the AM modulation depth for the VAR signal. The sum of the modulation depths for all enabled components must not exceed 100 %.

param vor_mod_depth
Range: 0 % to 100 %, Unit: %

6.15.2.1.4.6 IdSignal

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:ENABle
SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:MDEPth
SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:FREQuency
```

class IdSignalCls

IdSignal commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:ENABle
value: bool = driver.source.avionics.generator.vor.idSignal.get_enable()
```

Enables or disables the ID signal.

return
enable: OFF | ON

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:FREQuency
value: float = driver.source.avionics.generator.vor.idSignal.get_frequency()
```

Configures the audio frequency of the ID signal.

return
freq: Range: 0 Hz to 21 kHz, Unit: Hz

get_mod_depth() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:MDEPth
value: float = driver.source.avionics.generator.vor.idSignal.get_mod_depth()
```

Configures the modulation depth for the ID signal. The sum of the modulation depths for all enabled components must not exceed 100 %.

return
mod_depth: Range: 0 % to 100 %, Unit: %

set_enable(*enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:ENABle
driver.source.avionics.generator.vor.idSignal.set_enable(enable = False)
```

Enables or disables the ID signal.

param enable
OFF | ON

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:FREQuency
driver.source.avionics.generator.vor.idSignal.set_frequency(freq = 1.0)
```

Configures the audio frequency of the ID signal.

param freq
Range: 0 Hz to 21 kHz, Unit: Hz

set_mod_depth(*mod_depth: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:IDSignal:MDEPth
driver.source.avionics.generator.vor.idSignal.set_mod_depth(mod_depth = 1.0)
```

Configures the modulation depth for the ID signal. The sum of the modulation depths for all enabled components must not exceed 100 %.

param mod_depth
Range: 0 % to 100 %, Unit: %

6.15.2.1.4.7 RfSettings

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:LEVel
SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:FREQuency
```

class RfSettingsCls

RfSettings commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_frequency() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:FREQuency
value: float = driver.source.avionics.generator.vor.rfSettings.get_frequency()
```

Sets the center frequency of the unmodulated RF carrier.

return
freq: Range: 100 kHz to 3 GHz, Unit: Hz

get_level() → float

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:LEVel
value: float = driver.source.avionics.generator.vor.rfSettings.get_level()
```

Sets the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

return
level: Unit: dBm

set_frequency(*freq: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:FREQuency
driver.source.avionics.generator.vor.rfSettings.set_frequency(freq = 1.0)
```

Sets the center frequency of the unmodulated RF carrier.

param freq
Range: 100 kHz to 3 GHz, Unit: Hz

set_level(*level: float*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:LEVel
driver.source.avionics.generator.vor.rfSettings.set_level(level = 1.0)
```

Sets the RMS level of the unmodulated RF carrier. The allowed range depends on several other settings, for example on the selected connector, the frequency and the external attenuation. For supported output level ranges, refer to the data sheet.

param level
Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.avionics.generator.vor.rfSettings.clone()
```

Subgroups

6.15.2.1.4.8 RfOut

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:RFOut:ENABle
```

class RfOutCls

RfOut commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:RFOut:ENABle
value: bool = driver.source.avionics.generator.vor.rfSettings.rfOut.get_enable()
```

Enables or disables the RF output path.

return
rf_enable: OFF | ON

set_enable(*rf_enable: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:RFSettings:RFOut:ENABle
driver.source.avionics.generator.vor.rfSettings.rfOut.set_enable(rf_enable =␣
↪False)
```

Enables or disables the RF output path.

param rf_enable
OFF | ON

6.15.2.1.4.9 State

SCPI Command:

```
SOURce:AVIonics:GENerator<Instance>:VOR:STATe:ALL
SOURce:AVIonics:GENerator<Instance>:VOR:STATe
```

class StateCls

State commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get() → GeneratorState

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:STATe
value: enums.GeneratorState = driver.source.avionics.generator.vor.state.get()
```

Starts or stops the VOR generator or queries its state.

return
gen_state: OFF | ON | PENDing OFF Generator is off ON Generator is running PENDing Start or stop of generator is ongoing

get_all() → List[GeneratorState]

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:STATe:ALL
value: List[enums.GeneratorState] = driver.source.avionics.generator.vor.state.
↪get_all()
```

No command help available

return
all_states: No help available

set(*gen_control: bool*) → None

```
# SCPI: SOURce:AVIonics:GENerator<Instance>:VOR:STATe
driver.source.avionics.generator.vor.state.set(gen_control = False)
```

Starts or stops the VOR generator or queries its state.

param gen_control
ON | OFF ON Starts the generator OFF Stops the generator

6.15.3 Base

class BaseCls

Base commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.base.clone()
```

Subgroups

6.15.3.1 Adjustment

class AdjustmentCls

Adjustment commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.base.adjustment.clone()
```

Subgroups

6.15.3.1.1 State

SCPI Command:

```
SOURce:BASE:ADJustment:STATe
```

class StateCls

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → GeneratorState

```
# SCPI: SOURce:BASE:ADJustment:STATe
value: enums.GeneratorState = driver.source.base.adjustment.state.get()
```

Starts or terminates the adjustment of the reference frequency. A query returns the current state.

return
state: OFF | PENDing | ON OFF No adjustment in progress PENDing State transition ongoing ON Adjustment in progress

set(*control: bool*) → None

```
# SCPI: SOURce:BASE:ADJustment:STATe
driver.source.base.adjustment.state.set(control = False)
```

Starts or terminates the adjustment of the reference frequency. A query returns the current state.

param control
ON | OFF ON Starts the adjustment OFF Terminates the adjustment

6.15.4 Xrt

class XrtCls

Xrt commands group definition. 22 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.xrt.clone()
```

Subgroups

6.15.4.1 Generator

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:DSOurce
```

class GeneratorCls

Generator commands group definition. 22 total commands, 5 Subgroups, 1 group commands

get_dsource() → DigitalSource

```
# SCPI: SOURce:XRT:GENerator<Instance>:DSOurce
value: enums.DigitalSource = driver.source.xrt.generator.get_dsource()
```

Selects the data source for the XRT generator.

return
dsource: RF ARB | DMR | NXDN | POCSAG | P25 | ZigBee | DPMR

set_dsource(*dsource: DigitalSource*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:DSOurce
driver.source.xrt.generator.set_dsource(dsource = enums.DigitalSource.ARB)
```

Selects the data source for the XRT generator.

param dsource
RF ARB | DMR | NXDN | POCSAG | P25 | ZigBee | DPMR

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.xrt.generator.clone()
```

Subgroups

6.15.4.1.1 Arb

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:ARB:CRATe
SOURce:XRT:GENerator<Instance>:ARB:CRCProtect
SOURce:XRT:GENerator<Instance>:ARB:FOFFset
SOURce:XRT:GENerator<Instance>:ARB:LOFFset
SOURce:XRT:GENerator<Instance>:ARB:POFFset
SOURce:XRT:GENerator<Instance>:ARB:REPetition
```

class ArbCls

Arb commands group definition. 12 total commands, 2 Subgroups, 6 group commands

get_crate() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:CRATe
value: float = driver.source.xrt.generator.arb.get_crate()
```

Queries the clock rate of the loaded ARB file.

return
clock_rate: Unit: kHz

get_crc_protect() → YesNoStatus

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:CRCProtect
value: enums.YesNoStatus = driver.source.xrt.generator.arb.get_crc_protect()
```

Queries whether the loaded ARB file contains a CRC checksum.

return
crc_protection: NO | YES Unit: dB

get_foffset() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:FOFFset
value: float = driver.source.xrt.generator.arb.get_foffset()
```

Defines a frequency offset to be imposed at the baseband during ARB generation.

return
frequency_offset: Range: -10 MHz to 10 MHz, Unit: Hz

get_loffset() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:LOFFset
value: float = driver.source.xrt.generator.arb.get_loffset()
```

Queries the peak to average ratio (PAR) of the loaded ARB file. The PAR is also called level offset.

return
level_offset: Unit: dB

get_poffset() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:POFFset
value: float = driver.source.xrt.generator.arb.get_poffset()
```

Queries the peak offset of the loaded ARB file.

return
peak_offset: Unit: dB

get_repetition() → RepeatMode

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:REPetition
value: enums.RepeatMode = driver.source.xrt.generator.arb.get_repetition()
```

Defines how often the ARB file is processed.

return
repetition: CONTinuous | SINGle CONTinuous Cyclic continuous processing SINGle File is processed once.

set_foffset(*frequency_offset: float*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:FOFFset
driver.source.xrt.generator.arb.set_foffset(frequency_offset = 1.0)
```

Defines a frequency offset to be imposed at the baseband during ARB generation.

param frequency_offset
Range: -10 MHz to 10 MHz, Unit: Hz

set_repetition(*repetition: RepeatMode*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:REPetition
driver.source.xrt.generator.arb.set_repetition(repetition = enums.RepeatMode.
↪CONTinuous)
```

Defines how often the ARB file is processed.

param repetition
CONTinuous | SINGle CONTinuous Cyclic continuous processing SINGle File is processed once.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.xrt.generator.arb.clone()
```

Subgroups

6.15.4.1.1.1 File

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:ARB:FILE:DATE
SOURce:XRT:GENerator<Instance>:ARB:FILE:OPTion
SOURce:XRT:GENerator<Instance>:ARB:FILE:VERSion
SOURce:XRT:GENerator<Instance>:ARB:FILE
```

class FileCls

File commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_date() → str

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:FILE:DATE
value: str = driver.source.xrt.generator.arb.file.get_date()
```

Queries the date and time of the loaded ARB file.

return
date: String with date and time

get_option() → str

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:FILE:OPTion
value: str = driver.source.xrt.generator.arb.file.get_option()
```

Queries the options that are required to process the loaded ARB file.

return
options: String with comma-separated list of options.

get_value() → str

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:FILE
value: str = driver.source.xrt.generator.arb.file.get_value()
```

Selects the ARB file to be processed. Specify the path and the filename. If the file is stored in the folder corresponding to the @waveform alias, it is sufficient to specify only the filename.

INTRO_CMD_HELP: Example, the following strings are equivalent:

- 'D:/Rohde-Schwarz/CMA/Data/waveform/myfile.wv'
- '@WAVEFORM/myfile.wv'
- 'myfile.wv'

return
arb_file: String specifying the ARB file.

get_version() → str

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:FILE:VERSion
value: str = driver.source.xrt.generator.arb.file.get_version()
```

Queries the version of the loaded ARB file.

return
version: String containing the version. Empty string, if no file version is defined.

set_value(*arb_file: str*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:FILE
driver.source.xrt.generator.arb.file.set_value(arb_file = '1')
```

Selects the ARB file to be processed. Specify the path and the filename. If the file is stored in the folder corresponding to the @waveform alias, it is sufficient to specify only the filename.

INTRO_CMD_HELP: Example, the following strings are equivalent:

- 'D:/Rohde-Schwarz/CMA/Data/waveform/myfile.wv'
- '@WAVEFORM/myfile.wv'
- 'myfile.wv'

param arb_file
String specifying the ARB file.

6.15.4.1.1.2 Samples

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:ARB:SAMPles
```

class SamplesCls

Samples commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:SAMPles
value: float = driver.source.xrt.generator.arb.samples.get_value()
```

Queries the number of samples in the loaded ARB file.

return
samples: Range: 0 to 268173312

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.xrt.generator.arb.samples.clone()
```

Subgroups

6.15.4.1.1.3 Range

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:ARB:SAMPles:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class RangeStruct

Response structure. Fields:

- Range_Py: enums.ArbSamplesRange: FULL | SUB FULL Process all samples SUB Process a subrange according to Start and Stop.
- Start: int: Start of the subrange (always first sample, labeled zero) Range: 0 (fixed value)
- Stop: int: End of the subrange Range: 16 to samples in ARB file - 1

get() → RangeStruct

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:SAMPles:RANGe
value: RangeStruct = driver.source.xrt.generator.arb.samples.range.get()
```

Selects whether all samples or a subrange of samples is processed.

return
structure: for return value, see the help for RangeStruct structure arguments.

set(*range_py: ArbSamplesRange*, *start: int = None*, *stop: int = None*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:ARB:SAMPles:RANGe
driver.source.xrt.generator.arb.samples.range.set(range_py = enums.
↪ArbSamplesRange.FULL, start = 1, stop = 1)
```

Selects whether all samples or a subrange of samples is processed.

param range_py
FULL | SUB FULL Process all samples SUB Process a subrange according to Start and Stop.

param start
Start of the subrange (always first sample, labeled zero) Range: 0 (fixed value)

param stop
End of the subrange Range: 16 to samples in ARB file - 1

6.15.4.1.2 Digital

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:DIGital:FILE
```

class DigitalCls

Digital commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_file() → str

```
# SCPI: SOURce:XRT:GENerator<Instance>:DIGital:FILE
value: str = driver.source.xrt.generator.digital.get_file()
```

No command help available

return
arb_file: No help available

set_file(*arb_file: str*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:DIGital:FILE
driver.source.xrt.generator.digital.set_file(arb_file = '1')
```

No command help available

param arb_file
No help available

6.15.4.1.3 Reliability

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:RELiability
SOURce:XRT:GENerator<Instance>:RELiability:ALL
```

class ReliabilityCls

Reliability commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class AllStruct

Structure for reading output parameters. Fields:

- Reliability: int: See 'Reliability indicator values'
- Reliability_Msg: str: String indicating the error reason If there is no error (reliability value = 0) , the string is empty.
- Reliability_Add_Info: str: String providing additional information for an error. If there is no error (reliability value = 0) , the string is empty.

get(*details: str = None*) → str

```
# SCPI: SOURce:XRT:GENerator<Instance>:RELiability
value: str = driver.source.xrt.generator.reliability.get(details = '1')
```

Queries whether the generator has detected an error or not. If you have problems to generate a signal, use this command for troubleshooting. The returned parameters comprise a reliability indicator value and optionally, an error reason.

Use RsCma.reliability.last_value to read the updated reliability indicator.

param details
To return an error reason in addition to the reliability indicator value, append 'Details' to the query: SOUR:XRT:GEN:REL? 'DETails'

return
reliability_msg: String indicating the error reason. If there is no error (reliability value = 0) , the string is empty. The parameter is only returned if parameter Details = 'DETails'.

get_all() → AllStruct

```
# SCPI: SOURce:XRT:GENerator<Instance>:RELiability:ALL
value: AllStruct = driver.source.xrt.generator.reliability.get_all()
```

Queries whether the generator has detected an error or not. If you have problems to generate a signal, use this command for troubleshooting. The returned parameters comprise a reliability indicator value, an error reason and an additional information string.

return
structure: for return value, see the help for AllStruct structure arguments.

6.15.4.1.4 RfSettings

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:RFSettings:FREQuency
SOURce:XRT:GENerator<Instance>:RFSettings:LEVel
SOURce:XRT:GENerator<Instance>:RFSettings:PEPower
```

class RfSettingsCls

RfSettings commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_frequency() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:FREQuency
value: float = driver.source.xrt.generator.rfSettings.get_frequency()
```

Specifies the center frequency of the unmodulated RF carrier.

return
frequency: Range: 70 MHz to 6 GHz, Unit: Hz

get_level() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:LEVel
value: float = driver.source.xrt.generator.rfSettings.get_level()
```

Specifies the RMS level of the unmodulated RF signal.

return
level: Unit: dBm

get_pe_power() → float

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:PEPower
value: float = driver.source.xrt.generator.rfSettings.get_pe_power()
```

Queries the peak envelope power (PEP) .

return
pe_power: Unit: dBm

set_frequency(*frequency: float*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:FREQuency
driver.source.xrt.generator.rfSettings.set_frequency(frequency = 1.0)
```

Specifies the center frequency of the unmodulated RF carrier.

param frequency
Range: 70 MHz to 6 GHz, Unit: Hz

set_level(*level: float*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:LEVel
driver.source.xrt.generator.rfSettings.set_level(level = 1.0)
```

Specifies the RMS level of the unmodulated RF signal.

param level
Unit: dBm

set_pe_power(*pe_power: float*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:PEPower
driver.source.xrt.generator.rfSettings.set_pe_power(pe_power = 1.0)
```

Queries the peak envelope power (PEP) .

param pe_power
Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.xrt.generator.rfSettings.clone()
```

Subgroups

6.15.4.1.4.1 Connector<Connector>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.xrt.generator.rfSettings.connector.repcap_connector_get()
driver.source.xrt.generator.rfSettings.connector.repcap_connector_set(repcap.Connector.
↪Nr1)
```

class ConnectorCls

Connector commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Connector, default value after init: Connector.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.xrt.generator.rfSettings.connector.clone()
```

Subgroups

6.15.4.1.4.2 Enable

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:RFSettings:CONNector<nr>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*connector=Connector.Default*) → bool

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:CONNector<nr>:ENABle
value: bool = driver.source.xrt.generator.rfSettings.connector.enable.
↪get(connector = repcap.Connector.Default)
```

Activates the required output connectors from 'RF 1' to 'RF 8'.

param connector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

return
enable: OFF | ON

set(*enable: bool*, *connector=Connector.Default*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:RFSettings:CONNector<nr>:ENABle
driver.source.xrt.generator.rfSettings.connector.enable.set(enable = False,␣
↪connector = repcap.Connector.Default)
```

Activates the required output connectors from 'RF 1' to 'RF 8'.

param enable
OFF | ON

param connector
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

6.15.4.1.5 State

SCPI Command:

```
SOURce:XRT:GENerator<Instance>:STATe
SOURce:XRT:GENerator<Instance>:STATe:ALL
```

class StateCls

State commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get() → GeneratorState

```
# SCPI: SOURce:XRT:GENerator<Instance>:STATe
value: enums.GeneratorState = driver.source.xrt.generator.state.get()
```

Starts and stops the XRT100 generator.

return
gen_state: OFF | ON | PENDing OFF Generator is off. ON Generator is running. PENDing Start or stop of the generator is ongoing.

get_all() → List[GeneratorState]

```
# SCPI: SOURce:XRT:GENerator<Instance>:STATe:ALL
value: List[enums.GeneratorState] = driver.source.xrt.generator.state.get_all()
```

Queries all states of the XRT100 generator.

return
all_states: OFF | ON | PENDing

set(*gen_control: bool*) → None

```
# SCPI: SOURce:XRT:GENerator<Instance>:STATe
driver.source.xrt.generator.state.set(gen_control = False)
```

Starts and stops the XRT100 generator.

param gen_control
ON | OFF ON Starts the generator. OFF Stops the generator.

6.16 Status

SCPI Command:

```
STATus:PRESet
```

class StatusCls

Status commands group definition. 37 total commands, 7 Subgroups, 1 group commands

preset() → None

```
# SCPI: STATus:PRESet
driver.status.preset()
```

No command help available

preset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: STATus:PRESet
driver.status.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.clone()
```

Subgroups

6.16.1 Condition

class ConditionCls

Condition commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.condition.clone()
```

Subgroups

6.16.1.1 Bits

class BitsCls

Bits commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.condition.bits.clone()
```

Subgroups

6.16.1.1.1 All

SCPI Command:

```
STATus:CONDition:BITS:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → List[str]

```
# SCPI: STATus:CONDition:BITS:ALL
value: List[str] = driver.status.condition.bits.all.get(filter_py = '1', mode =␣
↪enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.16.1.1.2 Cataloge

SCPI Command:

```
STATus:CONDition:BITS:CATaloge
```

class CatalogeCls

Cataloge commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → List[str]

```
# SCPI: STATus:CONDition:BITS:CATaloge
value: List[str] = driver.status.condition.bits.cataloge.get(filter_py = '1',␣
↪mode = enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.16.1.1.3 Count

SCPI Command:

```
STATus:CONDition:BITS:COUNt
```

class CountCls

Count commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → int

```
# SCPI: STATus:CONDition:BITS:COUNt
value: int = driver.status.condition.bits.count.get(filter_py = '1', mode =␣
↪enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
count: No help available

6.16.2 Event

class EventCls

Event commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.event.clone()
```

Subgroups

6.16.2.1 Bits

SCPI Command:

```
STATus:EVENt:BITS:CLEar
```

class BitsCls

Bits commands group definition. 4 total commands, 3 Subgroups, 1 group commands

clear(*filter_py: str = None*, *mode: ExpressionMode = None*) → None

```
# SCPI: STATus:EVENt:BITS:CLEar
driver.status.event.bits.clear(filter_py = '1', mode = enums.ExpressionMode.
↪REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.event.bits.clone()
```

Subgroups

6.16.2.1.1 All

SCPI Command:

```
STATus:EVENt:BITS:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → List[str]

```
# SCPI: STATus:EVENt:BITS:ALL
value: List[str] = driver.status.event.bits.all.get(filter_py = '1', mode =␣
↪enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.16.2.1.2 Count

SCPI Command:

```
STATus:EVENt:BITS:COUNt
```

class CountCls

Count commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → int

```
# SCPI: STATus:EVENt:BITS:COUNt
value: int = driver.status.event.bits.count.get(filter_py = '1', mode = enums.
↪ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
count: No help available

6.16.2.1.3 Next

SCPI Command:

```
STATus:EVENt:BITS:NEXT
```

class NextCls

Next commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:EVENt:BITS:NEXT
value: str = driver.status.event.bits.next.get(filter_py = '1', mode = enums.
↪ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bit: No help available

6.16.3 Generator

class GeneratorCls

Generator commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.generator.clone()
```

Subgroups

6.16.3.1 Condition

class ConditionCls

Condition commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.generator.condition.clone()
```

Subgroups

6.16.3.1.1 Off

SCPI Command:

```
STATus:GENerator:CONDition:OFF
```

class OffCls

Off commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:GENerator:CONDition:OFF
value: str = driver.status.generator.condition.off.get(filter_py = '1', mode =␣
↪enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.3.1.2 On

SCPI Command:

```
STATus:GENerator:CONDition:ON
```

class OnCls

On commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:GENerator:CONDition:ON
value: str = driver.status.generator.condition.on.get(filter_py = '1', mode =␣
↪enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.3.1.3 Pending

SCPI Command:

```
STATus:GENerator:CONDition:PENDing
```

class PendingCls

Pending commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:GENerator:CONDition:PENDing
value: str = driver.status.generator.condition.pending.get(filter_py = '1',␣
↪mode = enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.4 Measurement

class MeasurementCls

Measurement commands group definition. 5 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.measurement.clone()
```

Subgroups

6.16.4.1 Condition

class ConditionCls

Condition commands group definition. 5 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.measurement.condition.clone()
```

Subgroups

6.16.4.1.1 Off

SCPI Command:

```
STATus:MEASurement:CONDition:OFF
```

class OffCls

Off commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:OFF
value: str = driver.status.measurement.condition.off.get(filter_py = '1', mode␣
↪= enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.4.1.2 Qued

SCPI Command:

```
STATus:MEASurement:CONDition:QUED
```

class QuedCls

Qued commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:QUED
value: str = driver.status.measurement.condition.qued.get(filter_py = '1', mode␣
↪= enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.4.1.3 Rdy

SCPI Command:

```
STATus:MEASurement:CONDition:RDY
```

class RdyCls

Rdy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:RDY
value: str = driver.status.measurement.condition.rdy.get(filter_py = '1', mode␣
↪= enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.4.1.4 Run

SCPI Command:

```
STATus:MEASurement:CONDition:RUN
```

class RunCls

Run commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:RUN
value: str = driver.status.measurement.condition.run.get(filter_py = '1', mode␣
↪= enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.4.1.5 SdReached

SCPI Command:

```
STATus:MEASurement:CONDition:SDReached
```

class SdReachedCls

SdReached commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*filter_py: str = None*, *mode: ExpressionMode = None*) → str

```
# SCPI: STATus:MEASurement:CONDition:SDReached
value: str = driver.status.measurement.condition.sdReached.get(filter_py = '1',␣
↪mode = enums.ExpressionMode.REGex)
```

No command help available

param filter_py
No help available

param mode
No help available

return
bitname: No help available

6.16.5 Operation

SCPI Command:

```
STATus:OPERation[:EVENt]
STATus:OPERation:CONDition
STATus:OPERation:ENABle
STATus:OPERation:PTRansition
STATus:OPERation:NTRansition
```

class OperationCls

Operation commands group definition. 10 total commands, 1 Subgroups, 5 group commands

get_condition() → int

```
# SCPI: STATus:OPERation:CONDition
value: int = driver.status.operation.get_condition()
```

No command help available

return
register_value: No help available

get_enable() → int

```
# SCPI: STATus:OPERation:ENABle
value: int = driver.status.operation.get_enable()
```

No command help available

return
register_value: No help available

get_event() → int

```
# SCPI: STATus:OPERation[:EVENt]
value: int = driver.status.operation.get_event()
```

No command help available

return
register_value: No help available

get_ntransition() → int

```
# SCPI: STATus:OPERation:NTRansition
value: int = driver.status.operation.get_ntransition()
```

No command help available

return
register_value: No help available

get_ptransition() → int

```
# SCPI: STATus:OPERation:PTRansition
value: int = driver.status.operation.get_ptransition()
```

No command help available

return
register_value: No help available

set_enable(*register_value: int*) → None

```
# SCPI: STATus:OPERation:ENABle
driver.status.operation.set_enable(register_value = 1)
```

No command help available

param register_value
No help available

set_ntransition(*register_value: int*) → None

```
# SCPI: STATus:OPERation:NTRansition
driver.status.operation.set_ntransition(register_value = 1)
```

No command help available

param register_value
No help available

set_ptransition(*register_value: int*) → None

```
# SCPI: STATus:OPERation:PTRansition
driver.status.operation.set_ptransition(register_value = 1)
```

No command help available

param register_value
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.operation.clone()
```

Subgroups

6.16.5.1 Bit<Bit>

RepCap Settings

```
# Range: Nr8 .. Nr12
rc = driver.status.operation.bit.repcap_bit_get()
driver.status.operation.bit.repcap_bit_set(repcap.Bit.Nr8)
```

class BitCls

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: Bit, default value after init: Bit.Nr8

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.operation.bit.clone()
```

Subgroups

6.16.5.1.1 Condition

SCPI Command:

```
STATus:OPERation:BIT<bitno>:CONDition
```

class ConditionCls

Condition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>:CONDition
value: bool = driver.status.operation.bit.condition.get(bit = repcap.Bit.
↪Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.16.5.1.2 Enable

SCPI Command:

```
STATus:OPERation:BIT<bitno>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → float

```
# SCPI: STATus:OPERation:BIT<bitno>:ENABle
value: float = driver.status.operation.bit.enable.get(bit = repcap.Bit.Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: float*, *bit=Bit.Default*) → None

```
# SCPI: STATus:OPERation:BIT<bitno>:ENABle
driver.status.operation.bit.enable.set(register_bit = 1.0, bit = repcap.Bit.
↪Default)
```

No command help available

param register_bit
No help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.16.5.1.3 Event

SCPI Command:

```
STATus:OPERation:BIT<bitno>[:EVENt]
```

class EventCls

Event commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>[:EVENt]
value: bool = driver.status.operation.bit.event.get(bit = repcap.Bit.Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.16.5.1.4 Ntransition

SCPI Command:

```
STATus:OPERation:BIT<bitno>:NTRansition
```

class NtransitionCls

Ntransition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>:NTRansition
value: bool = driver.status.operation.bit.ntransition.get(bit = repcap.Bit.
↪Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bit=Bit.Default*) → None

```
# SCPI: STATus:OPERation:BIT<bitno>:NTRansition
driver.status.operation.bit.ntransition.set(register_bit = False, bit = repcap.
↪Bit.Default)
```

No command help available

param register_bit
No help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.16.5.1.5 Ptransition

SCPI Command:

```
STATus:OPERation:BIT<bitno>:PTRansition
```

class PtransitionCls

Ptransition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:OPERation:BIT<bitno>:PTRansition
value: bool = driver.status.operation.bit.ptransition.get(bit = repcap.Bit.
↪Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bit=Bit.Default*) → None

```
# SCPI: STATus:OPERation:BIT<bitno>:PTRansition
driver.status.operation.bit.ptransition.set(register_bit = False, bit = repcap.
↪Bit.Default)
```

No command help available

param register_bit
No help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.16.6 Questionable

SCPI Command:

```
STATus:QUEStionable[:EVENt]
STATus:QUEStionable:CONDition
STATus:QUEStionable:ENABle
STATus:QUEStionable:PTRansition
STATus:QUEStionable:NTRansition
```

class QuestionableCls

Questionable commands group definition. 10 total commands, 1 Subgroups, 5 group commands

get_condition() → int

```
# SCPI: STATus:QUEStionable:CONDition
value: int = driver.status.questionable.get_condition()
```

No command help available

return
register_value: No help available

get_enable() → int

```
# SCPI: STATus:QUEStionable:ENABle
value: int = driver.status.questionable.get_enable()
```

No command help available

return
register_value: No help available

get_event() → int

```
# SCPI: STATus:QUEStionable[:EVENt]
value: int = driver.status.questionable.get_event()
```

No command help available

return
register_value: No help available

get_ntransition() → int

```
# SCPI: STATus:QUEStionable:NTRansition
value: int = driver.status.questionable.get_ntransition()
```

No command help available

return
register_value: No help available

get_ptransition() → int

```
# SCPI: STATus:QUEStionable:PTRansition
value: int = driver.status.questionable.get_ptransition()
```

No command help available

return
register_value: No help available

set_enable(*register_value: int*) → None

```
# SCPI: STATus:QUEStionable:ENABle
driver.status.questionable.set_enable(register_value = 1)
```

No command help available

param register_value
No help available

set_ntransition(*register_value: int*) → None

```
# SCPI: STATus:QUEStionable:NTRansition
driver.status.questionable.set_ntransition(register_value = 1)
```

No command help available

param register_value
No help available

set_ptransition(*register_value: int*) → None

```
# SCPI: STATus:QUEStionable:PTRansition
driver.status.questionable.set_ptransition(register_value = 1)
```

No command help available

param register_value
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.questionable.clone()
```

Subgroups

6.16.6.1 Bit<Bit>

RepCap Settings

```
# Range: Nr8 .. Nr12
rc = driver.status.questionable.bit.repcap_bit_get()
driver.status.questionable.bit.repcap_bit_set(repcap.Bit.Nr8)
```

class BitCls

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: Bit, default value after init: Bit.Nr8

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.status.questionable.bit.clone()
```

Subgroups

6.16.6.1.1 Condition

SCPI Command:

```
STATus:QUEStionable:BIT<bitno>:CONDition
```

class ConditionCls

Condition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:CONDition
value: bool = driver.status.questionable.bit.condition.get(bit = repcap.Bit.
↪Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.16.6.1.2 Enable

SCPI Command:

```
STATus:QUEStionable:BIT<bitno>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:ENABle
value: bool = driver.status.questionable.bit.enable.get(bit = repcap.Bit.
↪Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bit=Bit.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<bitno>:ENABle
driver.status.questionable.bit.enable.set(register_bit = False, bit = repcap.
↪Bit.Default)
```

No command help available

param register_bit
No help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.16.6.1.3 Event

SCPI Command:

```
STATus:QUEStionable:BIT<bitno>[:EVENt]
```

class EventCls

Event commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>[:EVENt]
value: bool = driver.status.questionable.bit.event.get(bit = repcap.Bit.Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

6.16.6.1.4 Ntransition

SCPI Command:

```
STATus:QUEStionable:BIT<bitno>:NTRansition
```

class NtransitionCls

Ntransition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:NTRansition
value: bool = driver.status.questionable.bit.ntransition.get(bit = repcap.Bit.
↪Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bit=Bit.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<bitno>:NTRansition
driver.status.questionable.bit.ntransition.set(register_bit = False, bit =␣
↪repcap.Bit.Default)
```

No command help available

param register_bit
No help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.16.6.1.5 Ptransition

SCPI Command:

```
STATus:QUEStionable:BIT<bitno>:PTRansition
```

class PtransitionCls

Ptransition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*bit=Bit.Default*) → bool

```
# SCPI: STATus:QUEStionable:BIT<bitno>:PTRansition
value: bool = driver.status.questionable.bit.ptransition.get(bit = repcap.Bit.
↪Default)
```

No command help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

return
register_bit: No help available

set(*register_bit: bool*, *bit=Bit.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<bitno>:PTRansition
driver.status.questionable.bit.ptransition.set(register_bit = False, bit =␣
↪repcap.Bit.Default)
```

No command help available

param register_bit
No help available

param bit
optional repeated capability selector. Default value: Nr8 (settable in the interface 'Bit')

6.16.7 Queue

SCPI Command:

```
STATus:QUEue[:NEXT]
```

class QueueCls

Queue commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class NextStruct

Structure for reading output parameters. Fields:

- Error_Code: int: No parameter help available
- Error_Description: str: No parameter help available

get_next() → NextStruct

```
# SCPI: STATus:QUEue[:NEXT]
value: NextStruct = driver.status.queue.get_next()
```

No command help available

return
structure: for return value, see the help for NextStruct structure arguments.

6.17 System

SCPI Command:

```
SYSTem:PRESet
SYSTem:PRESet:ALL
SYSTem:PRESet:BASE
SYSTem:RESet
SYSTem:RESet:ALL
SYSTem:RESet:BASE
```

class SystemCls

System commands group definition. 63 total commands, 9 Subgroups, 6 group commands

preset() → None

```
# SCPI: SYSTem:PRESet
driver.system.preset()
```

Presets or resets a selected application package in all scenarios. If <Application> is omitted, all applications are preset or reset.

preset_all() → None

```
# SCPI: SYSTem:PRESet:ALL
driver.system.preset_all()
```

Presets or resets the base settings and all applications in all scenarios.

preset_all_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:PRESet:ALL
driver.system.preset_all_with_opc()
```

Presets or resets the base settings and all applications in all scenarios.

Same as preset_all, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

preset_base() → None

```
# SCPI: SYSTem:PRESet:BASE
driver.system.preset_base()
```

Presets or resets only the base settings, not the applications. The method RsCma.System.presetBase and method RsCma. System.resetBase commands do not reset the settings for 'Start Automatically', 'Repetition' and 'Stop Condition' of the selftest configuration. See method RsCma.System.preset and method RsCma.System.reset.

preset_base_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:PRESet:BASE
driver.system.preset_base_with_opc()
```

Presets or resets only the base settings, not the applications. The method RsCma.System.presetBase and method RsCma. System.resetBase commands do not reset the settings for 'Start Automatically', 'Repetition' and 'Stop Condition' of the selftest configuration. See method RsCma.System.preset and method RsCma.System.reset.

Same as preset_base, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

preset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:PRESet
driver.system.preset_with_opc()
```

Presets or resets a selected application package in all scenarios. If <Application> is omitted, all applications are preset or reset.

Same as preset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

reset() → None

```
# SCPI: SYSTem:RESet
driver.system.reset()
```

Presets or resets a selected application package in all scenarios. If <Application> is omitted, all applications are preset or reset.

reset_all() → None

```
# SCPI: SYSTem:RESet:ALL
driver.system.reset_all()
```

Presets or resets the base settings and all applications in all scenarios.

reset_all_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:RESet:ALL
driver.system.reset_all_with_opc()
```

Presets or resets the base settings and all applications in all scenarios.

Same as reset_all, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

reset_base() → None

```
# SCPI: SYSTem:RESet:BASE
driver.system.reset_base()
```

Presets or resets only the base settings, not the applications. The method RsCma.System.presetBase and method RsCma. System.resetBase commands do not reset the settings for 'Start Automatically', 'Repetition' and 'Stop Condition' of the selftest configuration. See method RsCma.System.preset and method RsCma.System.reset.

reset_base_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:RESet:BASE
driver.system.reset_base_with_opc()
```

Presets or resets only the base settings, not the applications. The method RsCma.System.presetBase and method RsCma. System.resetBase commands do not reset the settings for 'Start Automatically', 'Repetition' and 'Stop Condition' of the selftest configuration. See method RsCma.System.preset and method RsCma.System.reset.

Same as reset_base, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:RESet
driver.system.reset_with_opc()
```

Presets or resets a selected application package in all scenarios. If <Application> is omitted, all applications are preset or reset.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.clone()
```

Subgroups

6.17.1 Base

SCPI Command:

```
SYSTem:BASE:RELiability
SYSTem:BASE:DID
SYSTem:BASE:KLOCk
SYSTem:BASE:VERSion
```

class BaseCls

Base commands group definition. 25 total commands, 11 Subgroups, 4 group commands

get_did() → str

```
# SCPI: SYSTem:BASE:DID
value: str = driver.system.base.get_did()
```

No command help available

return
device_id: No help available

get_klock() → bool

```
# SCPI: SYSTem:BASE:KLOCk
value: bool = driver.system.base.get_klock()
```

No command help available

return
klock: No help available

get_reliability() → int

```
# SCPI: SYSTem:BASE:RELiability
value: int = driver.system.base.get_reliability()
```

Returns a reliability value, indicating errors detected by the base software.

return
value: See 'Reliability indicator'

get_version() → float

```
# SCPI: SYSTem:BASE:VERSion
value: float = driver.system.base.get_version()
```

No command help available

return
version: No help available

set_klock(*klock: bool*) → None

```
# SCPI: SYSTem:BASE:KLOCk
driver.system.base.set_klock(klock = False)
```

No command help available

param klock
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.clone()
```

Subgroups

6.17.1.1 Date

SCPI Command:

```
SYSTem:BASE:DATE
```

class DateCls

Date commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DateStruct

Response structure. Fields:

- Year: int: No parameter help available
- Month: int: No parameter help available
- Day: int: No parameter help available

get() → DateStruct

```
# SCPI: SYSTem:BASE:DATE
value: DateStruct = driver.system.base.date.get()
```

No command help available

return
structure: for return value, see the help for DateStruct structure arguments.

set(*year: int*, *month: int*, *day: int*) → None

```
# SCPI: SYSTem:BASE:DATE
driver.system.base.date.set(year = 1, month = 1, day = 1)
```

No command help available

param year
No help available

param month
No help available

param day
No help available

6.17.1.2 Device

SCPI Command:

```
SYSTem:BASE:DEVice:ID
```

class DeviceCls

Device commands group definition. 3 total commands, 2 Subgroups, 1 group commands

get_id() → str

```
# SCPI: SYSTem:BASE:DEVice:ID
value: str = driver.system.base.device.get_id()
```

No command help available

return
device_id: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.device.clone()
```

Subgroups

6.17.1.2.1 License

SCPI Command:

```
SYSTem:BASE:DEVice:LICense
```

class LicenseCls

License commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LicenseStruct

Response structure. Fields:

- Sw_Option: List[str]: No parameter help available
- License_Count: List[int]: No parameter help available
- Instrument: List[int]: No parameter help available

get() → LicenseStruct

```
# SCPI: SYSTem:BASE:DEVice:LICense
value: LicenseStruct = driver.system.base.device.license.get()
```

No command help available

return
structure: for return value, see the help for LicenseStruct structure arguments.

set(*sw_option: List[str] = None*, *license_count: List[int] = None*, *instrument: List[int] = None*) → None

```
# SCPI: SYSTem:BASE:DEVice:LICense
driver.system.base.device.license.set(sw_option = ['1', '2', '3'], license_
↪count = [1, 2, 3], instrument = [1, 2, 3])
```

No command help available

param sw_option
No help available

param license_count
No help available

param instrument
No help available

6.17.1.2.2 Setup

SCPI Command:

```
SYSTem:BASE:DEVice:SETup
```

class SetupCls

Setup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SetupStruct

Response structure. Fields:

- Absolute_Item_Name: List[str]: No parameter help available
- Instrument: List[int]: No parameter help available

get() → SetupStruct

```
# SCPI: SYSTem:BASE:DEVice:SETup
value: SetupStruct = driver.system.base.device.setup.get()
```

No command help available

return
structure: for return value, see the help for SetupStruct structure arguments.

set(*absolute_item_name: List[str] = None*, *instrument: List[int] = None*) → None

```
# SCPI: SYSTem:BASE:DEVice:SETup
driver.system.base.device.setup.set(absolute_item_name = ['1', '2', '3'],␣
↪instrument = [1, 2, 3])
```

No command help available

param absolute_item_name
No help available

param instrument
No help available

6.17.1.3 Display

SCPI Command:

```
SYSTem:BASE:DISPlay:LANGuage
```

class DisplayCls

Display commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_language() → DisplayLanguage

```
# SCPI: SYSTem:BASE:DISPlay:LANGuage
value: enums.DisplayLanguage = driver.system.base.display.get_language()
```

Selects the GUI language to be used. The corresponding language and license must be installed.

return
language: AR | DE | SV | RU | KO | JA | EN | IT | FR | DA | ES | TR | CS | ZH

set_language(*language: DisplayLanguage*) → None

```
# SCPI: SYSTem:BASE:DISPlay:LANGuage
driver.system.base.display.set_language(language = enums.DisplayLanguage.AR)
```

Selects the GUI language to be used. The corresponding language and license must be installed.

param language
AR | DE | SV | RU | KO | JA | EN | IT | FR | DA | ES | TR | CS | ZH

6.17.1.4 Finish

SCPI Command:

```
SYSTem:BASE:FINish
```

class FinishCls

Finish commands group definition. 2 total commands, 1 Subgroups, 1 group commands

set() → None

```
# SCPI: SYSTem:BASE:FINish
driver.system.base.finish.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:BASE:FINish
driver.system.base.finish.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.finish.clone()
```

Subgroups

6.17.1.4.1 Device

SCPI Command:

```
SYSTem:BASE:FINish:DEVice
```

class DeviceCls

Device commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SYSTem:BASE:FINish:DEVice
driver.system.base.finish.device.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:BASE:FINish:DEVice
driver.system.base.finish.device.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.17.1.5 Gotsystem

SCPI Command:

```
SYSTem:BASE:GOTSystem
```

class GotsystemCls

Gotsystem commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SYSTem:BASE:GOTSystem
driver.system.base.gotsystem.set()
```

Minimizes the test software and shows the desktop of the operating system.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:BASE:GOTSystem
driver.system.base.gotsystem.set_with_opc()
```

Minimizes the test software and shows the desktop of the operating system.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.17.1.6 Option

class OptionCls

Option commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.option.clone()
```

Subgroups

6.17.1.6.1 ListPy

SCPI Command:

```
SYSTem:BASE:OPTion:LIST
```

class ListPyCls

ListPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*product_type: OptionsProductType = None*, *validity: OptionValidity = None*, *scope: OptionsScope = None*, *instrument_no: float = None*) → str

```
# SCPI: SYSTem:BASE:OPTion:LIST
value: str = driver.system.base.option.listPy.get(product_type = enums.
↪OptionsProductType.ALL, validity = enums.OptionValidity.ALL, scope = enums.
↪OptionsScope.INSTrument, instrument_no = 1.0)
```

Returns a list of installed software options (licenses) , hardware options, software packages and applications. The list can be filtered via parameters. If filtering results in an empty list, a '0' is returned.

param product_type
No help available

param validity
FUNCtional | VALid | ALL FUNCtional List only functional hardware options or applications. HWOPtion: Is functional if the hardware option and all its components can be used (no defect detected) . FWA: Is functional if the required hardware, software and license keys are available and functional. VALid List only valid software options. SWOPtion: Is valid if an active license key is available. ALL Disable filtering (default if parameter is omitted) .

param scope
No help available

param instrument_no
No help available

return
option_list: No help available

6.17.1.7 Password

SCPI Command:

```
SYSTem:BASE:PASSword:CDISable
```

class PasswordCls

Password commands group definition. 3 total commands, 1 Subgroups, 1 group commands

set_cdisable(*user_mode: UserRole*) → None

```
# SCPI: SYSTem:BASE:PASSword:CDISable
driver.system.base.password.set_cdisable(user_mode = enums.UserRole.ADMin)
```

No command help available

param user_mode
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.password.clone()
```

Subgroups

6.17.1.7.1 Cenable

SCPI Command:

```
SYSTem:BASE:PASSword[:CENable]:STATe
SYSTem:BASE:PASSword[:CENable]
```

class CenableCls

Cenable commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_state() → UserRole

```
# SCPI: SYSTem:BASE:PASSword[:CENable]:STATe
value: enums.UserRole = driver.system.base.password.cenable.get_state()
```

No command help available

return
user_mode: No help available

set(*user_mode: UserRole*, *password: str*) → None

```
# SCPI: SYSTem:BASE:PASSword[:CENable]
driver.system.base.password.cenable.set(user_mode = enums.UserRole.ADMin,␣
↪password = '1')
```

No command help available

param user_mode
No help available

param password
No help available

6.17.1.8 Reference

class ReferenceCls

Reference commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.reference.clone()
```

Subgroups

6.17.1.8.1 External

SCPI Command:

```
SYSTem:BASE:REFerence:EXTernal:LIRange
```

class ExternalCls

External commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_li_range() → LockRangeExternal

```
# SCPI: SYSTem:BASE:REFerence:EXTernal:LIRange
value: enums.LockRangeExternal = driver.system.base.reference.external.get_li_
↪range()
```

Selects the width of the lock-in range, used to synchronize to an external reference frequency source.

return
li_range: WIDE | MEDium | NARRow | INV INV means that the source is unusable, for example because of an ongoing adjustment

set_li_range(*li_range: LockRangeExternal*) → None

```
# SCPI: SYSTem:BASE:REFerence:EXTernal:LIRange
driver.system.base.reference.external.set_li_range(li_range = enums.
↪LockRangeExternal.INV)
```

Selects the width of the lock-in range, used to synchronize to an external reference frequency source.

param li_range
WIDE | MEDium | NARRow | INV INV means that the source is unusable, for example because of an ongoing adjustment

6.17.1.8.2 Frequency

SCPI Command:

```
SYSTem:BASE:REFerence:FREQuency:SOURce
```

class FrequencyCls

Frequency commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → RefFreqSource

```
# SCPI: SYSTem:BASE:REFerence:FREQuency:SOURce
value: enums.RefFreqSource = driver.system.base.reference.frequency.get_source()
```

Selects whether an internal or external reference frequency source is used.

return
source: INTernal | EXTernal

set_source(*source: RefFreqSource*) → None

```
# SCPI: SYSTem:BASE:REFerence:FREQuency:SOURce
driver.system.base.reference.frequency.set_source(source = enums.RefFreqSource.
↪EXTernal)
```

Selects whether an internal or external reference frequency source is used.

param source
INTernal | EXTernal

6.17.1.8.3 Internal

SCPI Command:

```
SYSTem:BASE:REFerence:INTernal:LIRange
```

class InternalCls

Internal commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_li_range() → LockRangeInternal

```
# SCPI: SYSTem:BASE:REFerence:INTernal:LIRange
value: enums.LockRangeInternal = driver.system.base.reference.internal.get_li_
↪range()
```

Selects the type of an internal reference frequency source.

return
li_range: MEDium | NARRow | INV MEDium TCXO NARRow OCXO INV Source unusable, for example adjustment ongoing

set_li_range(*li_range: LockRangeInternal*) → None

```
# SCPI: SYSTem:BASE:REFerence:INTernal:LIRange
driver.system.base.reference.internal.set_li_range(li_range = enums.
↪LockRangeInternal.INV)
```

Selects the type of an internal reference frequency source.

param li_range
MEDium | NARRow | INV MEDium TCXO NARRow OCXO INV Source unusable, for example adjustment ongoing

6.17.1.9 Restart

SCPI Command:

```
SYSTem:BASE:RESTart
```

class RestartCls

Restart commands group definition. 2 total commands, 1 Subgroups, 1 group commands

set() → None

```
# SCPI: SYSTem:BASE:RESTart
driver.system.base.restart.set()
```

Restarts the test software. This action is faster than a restart of the instrument and often sufficient.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:BASE:RESTart
driver.system.base.restart.set_with_opc()
```

Restarts the test software. This action is faster than a restart of the instrument and often sufficient.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.restart.clone()
```

Subgroups

6.17.1.9.1 Device

SCPI Command:

```
SYSTem:BASE:RESTart:DEVice
```

class DeviceCls

Device commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SYSTem:BASE:RESTart:DEVice
driver.system.base.restart.device.set()
```

Restarts the instrument.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:BASE:RESTart:DEVice
driver.system.base.restart.device.set_with_opc()
```

Restarts the instrument.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.17.1.10 Shutdown

SCPI Command:

```
SYSTem:BASE:SHUTdown
```

class ShutdownCls

Shutdown commands group definition. 2 total commands, 1 Subgroups, 1 group commands

set() → None

```
# SCPI: SYSTem:BASE:SHUTdown
driver.system.base.shutdown.set()
```

Shuts down the test software and shows the desktop of the operating system.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:BASE:SHUTdown
driver.system.base.shutdown.set_with_opc()
```

Shuts down the test software and shows the desktop of the operating system.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.shutdown.clone()
```

Subgroups

6.17.1.10.1 Device

SCPI Command:

```
SYSTem:BASE:SHUTdown:DEVice
```

class DeviceCls

Device commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: SYSTem:BASE:SHUTdown:DEVice
driver.system.base.shutdown.device.set()
```

Shuts down the instrument.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: SYSTem:BASE:SHUTdown:DEVice
driver.system.base.shutdown.device.set_with_opc()
```

Shuts down the instrument.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.17.1.11 Time

SCPI Command:

```
SYSTem:BASE:TIME
```

class TimeCls

Time commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class TimeStruct

Response structure. Fields:

- Hour: int: No parameter help available
- Min_Py: int: No parameter help available
- Sec: int: No parameter help available

get() → TimeStruct

```
# SCPI: SYSTem:BASE:TIME
value: TimeStruct = driver.system.base.time.get()
```

No command help available

return
structure: for return value, see the help for TimeStruct structure arguments.

set(*hour: int*, *min_py: int*, *sec: int*) → None

```
# SCPI: SYSTem:BASE:TIME
driver.system.base.time.set(hour = 1, min_py = 1, sec = 1)
```

No command help available

param hour
No help available

param min_py
No help available

param sec
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.base.time.clone()
```

Subgroups

6.17.1.11.1 Tzone

SCPI Command:

```
SYSTem:BASE:TIME:TZONe
```

class TzoneCls

Tzone commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TzoneStruct

Response structure. Fields:

- Hour: int: No parameter help available
- Minute: int: No parameter help available

get() → TzoneStruct

```
# SCPI: SYSTem:BASE:TIME:TZONe
value: TzoneStruct = driver.system.base.time.tzone.get()
```

No command help available

return
structure: for return value, see the help for TzoneStruct structure arguments.

set(*hour: int*, *minute: int*) → None

```
# SCPI: SYSTem:BASE:TIME:TZONe
driver.system.base.time.tzone.set(hour = 1, minute = 1)
```

No command help available

param hour
No help available

param minute
No help available

6.17.2 Communicate

class CommunicateCls

Communicate commands group definition. 18 total commands, 7 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.communicate.clone()
```

Subgroups

6.17.2.1 Gpib

SCPI Command:

```
SYSTem:COMMunicate:GPIB:VRESource
```

class GpibCls

Gpib commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_vresource() → str

```
# SCPI: SYSTem:COMMunicate:GPIB:VRESource
value: str = driver.system.communicate.gpib.get_vresource()
```

Queries the VISA resource string of the GPIB interface.

return
visa_resource: VISA resource string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.communicate.gpib.clone()
```

Subgroups

6.17.2.1.1 Self

SCPI Command:

```
SYSTem:COMMunicate:GPIB[:SELF]:ENABle
SYSTem:COMMunicate:GPIB[:SELF]:ADDR
```

class SelfCls

Self commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_addr() → int

```
# SCPI: SYSTem:COMMunicate:GPIB[:SELF]:ADDR
value: int = driver.system.communicate.gpib.self.get_addr()
```

Sets the primary GPIB address.

return
adress_no: GPIB address, integer number Range: 0 to 30

get_enable() → bool

```
# SCPI: SYSTem:COMMunicate:GPIB[:SELF]:ENABle
value: bool = driver.system.communicate.gpib.self.get_enable()
```

Enables or disables the GPIB interface.

return
enable: No help available

set_addr(*adress_no: int*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB[:SELF]:ADDR
driver.system.communicate.gpib.self.set_addr(adress_no = 1)
```

Sets the primary GPIB address.

param adress_no
GPIB address, integer number Range: 0 to 30

set_enable(*enable: bool*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB[:SELF]:ENABle
driver.system.communicate.gpib.self.set_enable(enable = False)
```

Enables or disables the GPIB interface.

param enable
1 | 0 1: GPIB enabled 0: GPIB disabled

6.17.2.2 Hislip

SCPI Command:

```
SYSTem:COMMunicate:HISLip:VRESource
```

class HislipCls

Hislip commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_vresource() → str

```
# SCPI: SYSTem:COMMunicate:HISLip:VRESource
value: str = driver.system.communicate.hislip.get_vresource()
```

Queries the VISA resource string for the HiSLIP protocol.

return
visa_resource: VISA resource string

6.17.2.3 Net

SCPI Command:

```
SYSTem:COMMunicate:NET:ADAPter
SYSTem:COMMunicate:NET:GATeway
SYSTem:COMMunicate:NET:IPADdress
SYSTem:COMMunicate:NET:HOSTname
SYSTem:COMMunicate:NET:DHCP
```

class NetCls

Net commands group definition. 7 total commands, 2 Subgroups, 5 group commands

get_adapter() → str

```
# SCPI: SYSTem:COMMunicate:NET:ADAPter
value: str = driver.system.communicate.net.get_adapter()
```

Selects the network adapter and thus the connection type to be modified. All SYSTem:COMMunicate:NET... commands affect the selected network adapter. This command does not activate or deactivate a network adapter.

return
network_adapter: No help available

get_dhcp() → bool

```
# SCPI: SYSTem:COMMunicate:NET:DHCP
value: bool = driver.system.communicate.net.get_dhcp()
```

Enables or disables DHCP.

return
dhcp_enable: No help available

get_gateway() → str

```
# SCPI: SYSTem:COMMunicate:NET:GATeway
value: str = driver.system.communicate.net.get_gateway()
```

Defines IPv4 addresses of default gateways. The configuration is only possible if DHCP is disabled. A query returns the currently defined addresses, irrespective of whether they have been specified manually or via DHCP.

return
gateway: No help available

get_hostname() → str

```
# SCPI: SYSTem:COMMunicate:NET:HOSTname
value: str = driver.system.communicate.net.get_hostname()
```

Sets the host name of the CMA.

return
hostname: No help available

get_ip_address() → List[str]

```
# SCPI: SYSTem:COMMunicate:NET:IPADdress
value: List[str] = driver.system.communicate.net.get_ip_address()
```

Assigns one or more IPv4 addresses to the network adapter. The configuration is only possible if DHCP is disabled. A query returns the currently assigned addresses, irrespective of whether they have been assigned manually or via DHCP.

return
ip_address: No help available

set_adapter(*network_adapter: str*) → None

```
# SCPI: SYSTem:COMMunicate:NET:ADAPter
driver.system.communicate.net.set_adapter(network_adapter = '1')
```

Selects the network adapter and thus the connection type to be modified. All SYSTem:COMMunicate:NET... commands affect the selected network adapter. This command does not activate or deactivate a network adapter.

param network_adapter
String parameter

set_dhcp(*dhcp_enable: bool*) → None

```
# SCPI: SYSTem:COMMunicate:NET:DHCP
driver.system.communicate.net.set_dhcp(dhcp_enable = False)
```

Enables or disables DHCP.

param dhcp_enable
1 | 0 1: DHCP enabled and automatic TCP/IP address configuration 0: DHCP disabled and manual address configuration

set_gateway(*gateway: str*) → None

```
# SCPI: SYSTem:COMMunicate:NET:GATeway
driver.system.communicate.net.set_gateway(gateway = '1')
```

Defines IPv4 addresses of default gateways. The configuration is only possible if DHCP is disabled. A query returns the currently defined addresses, irrespective of whether they have been specified manually or via DHCP.

param gateway
String parameter, gateway IP address consisting of four blocks separated by dots Several strings separated by commas can be entered, or several addresses separated by commas can be included in one string.

set_hostname(*hostname: str*) → None

```
# SCPI: SYSTem:COMMunicate:NET:HOSTname
driver.system.communicate.net.set_hostname(hostname = '1')
```

Sets the host name of the CMA.

param hostname
Host name as string

set_ip_address(*ip_address: List[str]*) → None

```
# SCPI: SYSTem:COMMunicate:NET:IPADdress
driver.system.communicate.net.set_ip_address(ip_address = ['1', '2', '3'])
```

Assigns one or more IPv4 addresses to the network adapter. The configuration is only possible if DHCP is disabled. A query returns the currently assigned addresses, irrespective of whether they have been assigned manually or via DHCP.

param ip_address
String parameter, IP address consisting of four blocks (octets) separated by dots Several strings separated by commas can be entered, or several addresses separated by commas can be included in one string.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.communicate.net.clone()
```

Subgroups

6.17.2.3.1 Dns

SCPI Command:

```
SYSTem:COMMunicate:NET:DNS:ENABle
```

class DnsCls

Dns commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: SYSTem:COMMunicate:NET:DNS:ENABle
value: bool = driver.system.communicate.net.dns.get_enable()
```

Enables or disables the configuration of DNS addresses via DHCP.

return
dns_enable: No help available

set_enable(*dns_enable: bool*) → None

```
# SCPI: SYSTem:COMMunicate:NET:DNS:ENABle
driver.system.communicate.net.dns.set_enable(dns_enable = False)
```

Enables or disables the configuration of DNS addresses via DHCP.

param dns_enable
1 | 0 1: Enabled, automatic address configuration 0: Disabled, manual address configuration

6.17.2.3.2 Subnet

SCPI Command:

```
SYSTem:COMMunicate:NET:SUBNet:MASK
```

class SubnetCls

Subnet commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mask() → List[str]

```
# SCPI: SYSTem:COMMunicate:NET:SUBNet:MASK
value: List[str] = driver.system.communicate.net.subnet.get_mask()
```

Defines the subnet masks to be used for the network adapter. The configuration is only possible if DHCP is disabled. A query returns the currently used subnet masks, irrespective of whether they have been assigned manually or via DHCP.

return
subnet_mask: No help available

set_mask(*subnet_mask: List[str]*) → None

```
# SCPI: SYSTem:COMMunicate:NET:SUBNet:MASK
driver.system.communicate.net.subnet.set_mask(subnet_mask = ['1', '2', '3'])
```

Defines the subnet masks to be used for the network adapter. The configuration is only possible if DHCP is disabled. A query returns the currently used subnet masks, irrespective of whether they have been assigned manually or via DHCP.

param subnet_mask
String parameter, subnet mask consisting of four blocks separated by dots Several strings separated by commas can be entered, or several masks separated by commas can be included in one string.

6.17.2.4 Rsib

SCPI Command:

```
SYSTem:COMMunicate:RSIB:VRESource
```

class RsibCls

Rsib commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_vresource() → str

```
# SCPI: SYSTem:COMMunicate:RSIB:VRESource
value: str = driver.system.communicate.rsib.get_vresource()
```

No command help available

return
visa_resource: No help available

6.17.2.5 Socket

SCPI Command:

```
SYSTem:COMMunicate:SOCKet:VRESource
SYSTem:COMMunicate:SOCKet:MODE
SYSTem:COMMunicate:SOCKet:PORT
```

class SocketCls

Socket commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_mode() → ProtocolMode

```
# SCPI: SYSTem:COMMunicate:SOCKet:MODE
value: enums.ProtocolMode = driver.system.communicate.socket.get_mode()
```

Sets the protocol operation mode for direct socket communication.

return
protocol_mode: No help available

get_port() → int

```
# SCPI: SYSTem:COMMunicate:SOCKet:PORT
value: int = driver.system.communicate.socket.get_port()
```

Sets the data port number for direct socket communication.

return
port_number: No help available

get_vresource() → str

```
# SCPI: SYSTem:COMMunicate:SOCKet:VRESource
value: str = driver.system.communicate.socket.get_vresource()
```

Queries the VISA resource string of the socket resource (direct socket communication) .

return
visa_resource: VISA resource string

set_mode(*protocol_mode: ProtocolMode*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:MODE
driver.system.communicate.socket.set_mode(protocol_mode = enums.ProtocolMode.
↪AGILent)
```

Sets the protocol operation mode for direct socket communication.

param protocol_mode
RAW | AGILent | IEEE1174 RAW No support of control messages AGILent Emulation codes via control connection (control port) IEEE1174 Emulation codes via data connection (data port)

set_port(*port_number: int*) → None

```
# SCPI: SYSTem:COMMunicate:SOCKet:PORT
driver.system.communicate.socket.set_port(port_number = 1)
```

Sets the data port number for direct socket communication.

param port_number
To select a free port number, enter 0. To select a specific port number, use the following range. Range: 1024 to 32767

6.17.2.6 Usb

SCPI Command:

```
SYSTem:COMMunicate:USB:VRESource
```

class UsbCls

Usb commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_vresource() → str

```
# SCPI: SYSTem:COMMunicate:USB:VRESource
value: str = driver.system.communicate.usb.get_vresource()
```

Queries the VISA resource string of the USB interface.

return
visa_resource: VISA resource string

6.17.2.7 Vxi

SCPI Command:

```
SYSTem:COMMunicate:VXI:VRESource
SYSTem:COMMunicate:VXI:GTR
```

class VxiCls

Vxi commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_gtr() → bool

```
# SCPI: SYSTem:COMMunicate:VXI:GTR
value: bool = driver.system.communicate.vxi.get_gtr()
```

No command help available

return
bool_switchremote: No help available

get_vresource() → str

```
# SCPI: SYSTem:COMMunicate:VXI:VRESource
value: str = driver.system.communicate.vxi.get_vresource()
```

Queries the VISA resource string for the VXI-11 protocol.

return
visa_resource: VISA resource string

set_gtr(*bool_switchremote: bool*) → None

```
# SCPI: SYSTem:COMMunicate:VXI:GTR
driver.system.communicate.vxi.set_gtr(bool_switchremote = False)
```

No command help available

param bool_switchremote
No help available

6.17.3 DeviceFootprint

SCPI Command:

```
SYSTem:DFPRint
```

class DeviceFootprintCls

DeviceFootprint commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get() → bytes

```
# SCPI: SYSTem:DFPRint
value: bytes = driver.system.deviceFootprint.get()
```

No command help available

return
xml_device_footprint: No help available

set(*path: str = None*) → None

```
# SCPI: SYSTem:DFPRint
driver.system.deviceFootprint.set(path = '1')
```

No command help available

param path
No help available

6.17.4 Display

SCPI Command:

```
SYSTem:DISPlay:UPDate
```

class DisplayCls

Display commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_update() → bool

```
# SCPI: SYSTem:DISPlay:UPDate
value: bool = driver.system.display.get_update()
```

Defines whether the display is updated or not while the instrument is in the remote state. Disabling the update speeds up testing and is the recommended state. See also 'Using the display during remote control'.

return
display_update: No help available

set_update(*display_update: bool*) → None

```
# SCPI: SYSTem:DISPlay:UPDate
driver.system.display.set_update(display_update = False)
```

Defines whether the display is updated or not while the instrument is in the remote state. Disabling the update speeds up testing and is the recommended state. See also 'Using the display during remote control'.

param display_update
1 | 0 1: The display is shown and updated during remote control. 0: The display shows static image during remote control.

6.17.5 Error

SCPI Command:

```
SYSTem:ERRor:ALL
SYSTem:ERRor:COUNt
```

class ErrorCls

Error commands group definition. 4 total commands, 1 Subgroups, 2 group commands

class AllStruct

Structure for reading output parameters. Fields:

- Error_Number: int: 0 means that the queue is empty. Positive error codes are instrument-dependent. Negative error codes are reserved by the SCPI standard.
- Error_Text: str: String specifying the error

get_all() → AllStruct

```
# SCPI: SYSTem:ERRor:ALL
value: AllStruct = driver.system.error.get_all()
```

Queries and deletes all entries in the error queue. Each entry consists of an error code and a short description of the error. The entries are returned as comma-separated list: {<ErrorNumber>, <ErrorText>}entry 1, {<ErrorNumber>, <ErrorText>}entry 2, ..., {<ErrorNumber>, <ErrorText>}entry n

return
structure: for return value, see the help for AllStruct structure arguments.

get_count() → int

```
# SCPI: SYSTem:ERRor:COUNt
value: int = driver.system.error.get_count()
```

Queries the number of entries in the error queue.

return
error_count: Number of entries Range: 0 to n

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.error.clone()
```

Subgroups

6.17.5.1 Code

SCPI Command:

```
SYSTem:ERRor:CODE:ALL
SYSTem:ERRor:CODE[:NEXT]
```

class CodeCls

Code commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_all() → int

```
# SCPI: SYSTem:ERRor:CODE:ALL
value: int = driver.system.error.code.get_all()
```

Queries the error code numbers of all entries in the error queue and deletes all entries.

return
error_code: Comma-separated list of error codes 0 means that the queue is empty. Positive error codes are instrument-dependent. Negative error codes are reserved by the SCPI standard.

get_next() → int

```
# SCPI: SYSTem:ERRor:CODE[:NEXT]
value: int = driver.system.error.code.get_next()
```

Queries the code number of the oldest entry in the error queue and deletes the entry.

return
error: 0 means that the queue is empty. Positive error codes are instrument-dependent. Negative error codes are reserved by the SCPI standard.

6.17.6 Help

class HelpCls

Help commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.help.clone()
```

Subgroups

6.17.6.1 Headers

SCPI Command:

```
SYSTem:HELP:HEADers
```

class HeadersCls

Headers commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*parser: str = None*) → bytes

```
# SCPI: SYSTem:HELP:HEADers
value: bytes = driver.system.help.headers.get(parser = '1')
```

No command help available

param parser
No help available

return
headers: No help available

6.17.6.2 Status

SCPI Command:

```
SYSTem:HELP:STATus:BITS
SYSTem:HELP:STATus[:REGister]
```

class StatusCls

Status commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_bits() → List[str]

```
# SCPI: SYSTem:HELP:STATus:BITS
value: List[str] = driver.system.help.status.get_bits()
```

No command help available

return
bits: No help available

get_register() → List[str]

```
# SCPI: SYSTem:HELP:STATus[:REGister]
value: List[str] = driver.system.help.status.get_register()
```

No command help available

return
register: No help available

6.17.6.3 Syntax

SCPI Command:

```
SYSTem:HELP:SYNTax
SYSTem:HELP:SYNTax:ALL
```

class SyntaxCls

Syntax commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get(*header: str*) → bytes

```
# SCPI: SYSTem:HELP:SYNTax
value: bytes = driver.system.help.syntax.get(header = '1')
```

No command help available

param header
No help available

return
syntax: No help available

get_all() → bytes

```
# SCPI: SYSTem:HELP:SYNTax:ALL
value: bytes = driver.system.help.syntax.get_all()
```

No command help available

return
syntax: No help available

6.17.7 Option

class OptionCls

Option commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.option.clone()
```

Subgroups

6.17.7.1 Version

SCPI Command:

```
SYSTem:OPTion:VERSion
```

class VersionCls

Version commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*applicname: str = None*) → str

```
# SCPI: SYSTem:OPTion:VERSion
value: str = driver.system.option.version.get(applicname = '1')
```

Returns version information for installed software packages.
INTRO_CMD_HELP: The structure of the returned string depends on the parameter <Application>:

- If <Application> is specified Returned string: '<Version>' '0' means that the application is unknown or not installed.
- If <Application> is omitted Returned string: '<PackageName1>,<Version1>;<PackageName2>,<Version2>;...'

param applicname
No help available

return
option_list: No help available

6.17.8 Password

class PasswordCls

Password commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.system.password.clone()
```

Subgroups

6.17.8.1 New

SCPI Command:

```
SYSTem:PASSword:NEW
```

class NewCls

New commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*current_password: str*, *new_password: str*) → None

```
# SCPI: SYSTem:PASSword:NEW
driver.system.password.new.set(current_password = '1', new_password = '1')
```

No command help available

param current_password
No help available

param new_password
No help available

6.17.9 Update

SCPI Command:

```
SYSTem:UPDate:DGRoup
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_dgroup() → str

```
# SCPI: SYSTem:UPDate:DGRoup
value: str = driver.system.update.get_dgroup()
```

No command help available

return
devicegroup: No help available

set_dgroup(*devicegroup: str*) → None

```
# SCPI: SYSTem:UPDate:DGRoup
driver.system.update.set_dgroup(devicegroup = '1')
```

No command help available

param devicegroup
No help available

6.18 Trace

class TraceCls

Trace commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.clone()
```

Subgroups

6.18.1 Remote

class RemoteCls

Remote commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.remote.clone()
```

Subgroups

6.18.1.1 Mode

class ModeCls

Mode commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.remote.mode.clone()
```

Subgroups

6.18.1.1.1 Display

SCPI Command:

```
TRACe:REMote:MODE:DISPlay:CLEar
```

class DisplayCls

Display commands group definition. 1 total commands, 0 Subgroups, 1 group commands

clear() → None

```
# SCPI: TRACe:REMote:MODE:DISPlay:CLEar
driver.trace.remote.mode.display.clear()
```

Clears the display of the SCPI remote trace in analysis mode.

clear_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: TRACe:REMote:MODE:DISPlay:CLEar
driver.trace.remote.mode.display.clear_with_opc()
```

Clears the display of the SCPI remote trace in analysis mode.

Same as clear, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.18.1.1.2 File<Instrument>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.trace.remote.mode.file.repcap_instrument_get()
driver.trace.remote.mode.file.repcap_instrument_set(repcap.Instrument.Nr1)
```

class FileCls

File commands group definition. 7 total commands, 7 Subgroups, 0 group commands Repeated Capability: Instrument, default value after init: Instrument.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trace.remote.mode.file.clone()
```

Subgroups

6.18.1.1.2.1 Enable

SCPI Command:

```
TRACe:REMote:MODE:FILE<instrument>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*instrument=Instrument.Default*) → bool

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:ENABle
value: bool = driver.trace.remote.mode.file.enable.get(instrument = repcap.
↪Instrument.Default)
```

Enables or disables tracing of the remote control interface to a file.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

return
benable: No help available

set(*benable: bool*, *instrument=Instrument.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:ENABle
driver.trace.remote.mode.file.enable.set(benable = False, instrument = repcap.
↪Instrument.Default)
```

Enables or disables tracing of the remote control interface to a file.

param benable
1 | 0 1: Tracing is enabled. 0: Tracing is disabled. Default value: 0

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

6.18.1.1.2.2 FilterPy

SCPI Command:

```
TRACe:REMote:MODE:FILE<instrument>:FILTer
```

class FilterPyCls

FilterPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FilterPyStruct

Structure for setting input parameters. Fields:

- Binput: bool: No parameter help available
- Boutput: bool: No parameter help available
- Berror: bool: No parameter help available
- Btrigger: bool: No parameter help available
- Bdevice_Clear: bool: No parameter help available
- Bstatus_Register: bool: No parameter help available
- Bconnection: bool: No parameter help available
- Bremote_Local_Events: bool: No parameter help available

get(*instrument=Instrument.Default*) → FilterPyStruct

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FILTer
value: FilterPyStruct = driver.trace.remote.mode.file.filterPy.get(instrument =␣
↪repcap.Instrument.Default)
```

Specifies a filter for tracing of the remote control interface. The filter defines which message types and events are traced to a file.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

return
structure: for return value, see the help for FilterPyStruct structure arguments.

set(*structure: FilterPyStruct*, *instrument=Instrument.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FILTer
structure = driver.trace.remote.mode.file.filterPy.FilterPyStruct()
structure.Binput: bool = False
structure.Boutput: bool = False
structure.Berror: bool = False
structure.Btrigger: bool = False
structure.Bdevice_Clear: bool = False
structure.Bstatus_Register: bool = False
structure.Bconnection: bool = False
structure.Bremote_Local_Events: bool = False
driver.trace.remote.mode.file.filterPy.set(structure, instrument = repcap.
↪Instrument.Default)
```

Specifies a filter for tracing of the remote control interface. The filter defines which message types and events are traced to a file.

param structure
for set value, see the help for FilterPyStruct structure arguments.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

6.18.1.1.2.3 FormatPy

SCPI Command:

```
TRACe:REMote:MODE:FILE<instrument>:FORMat
```

class FormatPyCls

FormatPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*instrument=Instrument.Default*) → Eformat

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FORMat
value: enums.Eformat = driver.trace.remote.mode.file.formatPy.get(instrument =␣
↪repcap.Instrument.Default)
```

Specifies the target file format for tracing of the remote control interface. The trace can be stored as ASCII file or as XML file.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

return
eformat: No help available

set(*eformat: Eformat*, *instrument=Instrument.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:FORMat
driver.trace.remote.mode.file.formatPy.set(eformat = enums.Eformat.ASCii,␣
↪instrument = repcap.Instrument.Default)
```

Specifies the target file format for tracing of the remote control interface. The trace can be stored as ASCII file or as XML file.

param eformat
ASCii | XML Default value: XML

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

6.18.1.1.2.4 Name

SCPI Command:

```
TRACe:REMote:MODE:FILE<instrument>:NAME
```

class NameCls

Name commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*instrument=Instrument.Default*) → str

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:NAME
value: str = driver.trace.remote.mode.file.name.get(instrument = repcap.
↪Instrument.Default)
```

Specifies path and name of the target file for tracing of the remote control interface. If you specify a new target file while tracing, the old target file is closed, the new file is created and tracing is continued with the new file.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

return
bs_file_path: No help available

set(*bs_file_path: str*, *instrument=Instrument.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:NAME
driver.trace.remote.mode.file.name.set(bs_file_path = '1', instrument = repcap.
↪Instrument.Default)
```

Specifies path and name of the target file for tracing of the remote control interface. If you specify a new target file while tracing, the old target file is closed, the new file is created and tracing is continued with the new file.

param bs_file_path
String parameter specifying path and name of the file Default value: 'D:/Rohde-Schwarz/CMA/Log/version/RemoteTrace-Inst0.xml'

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

6.18.1.1.2.5 Size

SCPI Command:

```
TRACe:REMote:MODE:FILE<instrument>:SIZE
```

class SizeCls

Size commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*instrument=Instrument.Default*) → int

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:SIZE
value: int = driver.trace.remote.mode.file.size.get(instrument = repcap.
↪Instrument.Default)
```

Specifies the maximum trace file size in bytes.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

return
ifile_size: No help available

set(*ifile_size: int*, *instrument=Instrument.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:SIZE
driver.trace.remote.mode.file.size.set(ifile_size = 1, instrument = repcap.
↪Instrument.Default)
```

Specifies the maximum trace file size in bytes.

param ifile_size
Recommended minimum value: 40000 bytes Maximum value: 1000000000 bytes (1 GB) Default value: 1000000000 bytes (1 GB)

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

6.18.1.1.2.6 StartMode

SCPI Command:

```
TRACe:REMote:MODE:FILE<instrument>:STARtmode
```

class StartModeCls

StartMode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*instrument=Instrument.Default*) → EstartMode

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STARtmode
value: enums.EstartMode = driver.trace.remote.mode.file.startMode.
↪get(instrument = repcap.Instrument.Default)
```

Specifies whether tracing is started automatically or manually.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

return
estart_mode: AUTO | EXPLicit AUTO Start tracing automatically when the instrument is started. EXPLicit Start tracing via the command method RsCma.Trace.Remote.Mode.File.Enable.set. Default value: EXPLicit

set(*estart_mode: EstartMode*, *instrument=Instrument.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STARtmode
driver.trace.remote.mode.file.startMode.set(estart_mode = enums.EstartMode.AUTO,
↪ instrument = repcap.Instrument.Default)
```

Specifies whether tracing is started automatically or manually.

param estart_mode
AUTO | EXPLicit AUTO Start tracing automatically when the instrument is started. EXPLicit Start tracing via the command method RsCma.Trace.Remote.Mode.File.Enable.set. Default value: EXPLicit

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

6.18.1.1.2.7 StopMode

SCPI Command:

```
TRACe:REMote:MODE:FILE<instrument>:STOPmode
```

class StopModeCls

StopMode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*instrument=Instrument.Default*) → EstopMode

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STOPmode
value: enums.EstopMode = driver.trace.remote.mode.file.stopMode.get(instrument␣
↪= repcap.Instrument.Default)
```

Specifies how / when tracing is stopped and the trace file is closed.

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

return
estop_mode: AUTO | EXPLicit | ERRor | BUFFerfull AUTO Stop tracing automatically when the instrument is shut down. EXPLicit Stop tracing via method RsCma.Trace.Remote.Mode.File.Enable.set. ERRor Stop tracing when a SCPI error occurs. BUFFerfull Stop tracing when the maximum file size is reached. Default value: EXPLicit

set(*estop_mode: EstopMode*, *instrument=Instrument.Default*) → None

```
# SCPI: TRACe:REMote:MODE:FILE<instrument>:STOPmode
driver.trace.remote.mode.file.stopMode.set(estop_mode = enums.EstopMode.AUTO,␣
↪instrument = repcap.Instrument.Default)
```

Specifies how / when tracing is stopped and the trace file is closed.

param estop_mode
AUTO | EXPLicit | ERRor | BUFFerfull AUTO Stop tracing automatically when the instrument is shut down. EXPLicit Stop tracing via method

RsCma.Trace.Remote.Mode.File.Enable.set. ERRor Stop tracing when a SCPI error occurs. BUFFerfull Stop tracing when the maximum file size is reached. Default value: EXPLicit

param instrument
optional repeated capability selector. Default value: Nr1 (settable in the interface 'File')

6.19 Trigger

class TriggerCls

Trigger commands group definition. 74 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.clone()
```

Subgroups

6.19.1 AfRf

class AfRfCls

AfRf commands group definition. 41 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.clone()
```

Subgroups

6.19.1.1 Generator

class GeneratorCls

Generator commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.generator.clone()
```

Subgroups

6.19.1.1.1 Arb

SCPI Command:

```
TRIGger:AFRF:GENerator<Instance>:ARB:AUTostart
TRIGger:AFRF:GENerator<Instance>:ARB:DELay
TRIGger:AFRF:GENerator<Instance>:ARB:RETRigger
TRIGger:AFRF:GENerator<Instance>:ARB:SOURce
```

class ArbCls

Arb commands group definition. 6 total commands, 2 Subgroups, 4 group commands

get_autostart() → bool

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:AUTostart
value: bool = driver.trigger.afRf.generator.arb.get_autostart()
```

Enables or disables the automatic start of the loaded ARB file whenever the generator is turned on. This setting applies only to the 'Manual' trigger source. For other trigger sources, autostart is disabled.

return
autostart: OFF | ON

get_delay() → float

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:DELay
value: float = driver.trigger.afRf.generator.arb.get_delay()
```

Specifies a start delay relative to the trigger event. This setting is ignored for the 'Manual' trigger source.

return
delay: Range: 0 s to 100 s, Unit: s

get_retrigger() → bool

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:RETRigger
value: bool = driver.trigger.afRf.generator.arb.get_retrigger()
```

Specifies whether trigger events during ARB file processing restart the ARB file or not. This setting applies only to the 'Manual' trigger source. For other trigger sources, retriggering is disabled.

return
retrigger: OFF | ON

get_source() → str

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:SOURce
value: str = driver.trigger.afRf.generator.arb.get_source()
```

Selects the trigger event source used to start or restart ARB file processing. To query a list of all supported sources, use method RsCma.Trigger.AfRf.Generator.Arb.Catalog.source.

return
source: 'Manual' Manual start via method RsCma.Trigger.AfRf.Generator.Arb.Manual.Execute.set
'Base1: External TRIG In' External trigger signal received at connector TRIG IN

set_autostart(*autostart: bool*) → None

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:AUTostart
driver.trigger.afRf.generator.arb.set_autostart(autostart = False)
```

Enables or disables the automatic start of the loaded ARB file whenever the generator is turned on. This setting applies only to the 'Manual' trigger source. For other trigger sources, autostart is disabled.

param autostart
OFF | ON

set_delay(*delay: float*) → None

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:DELay
driver.trigger.afRf.generator.arb.set_delay(delay = 1.0)
```

Specifies a start delay relative to the trigger event. This setting is ignored for the 'Manual' trigger source.

param delay
Range: 0 s to 100 s, Unit: s

set_retrigger(*retrigger: bool*) → None

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:RETRigger
driver.trigger.afRf.generator.arb.set_retrigger(retrigger = False)
```

Specifies whether trigger events during ARB file processing restart the ARB file or not. This setting applies only to the 'Manual' trigger source. For other trigger sources, retriggering is disabled.

param retrigger
OFF | ON

set_source(*source: str*) → None

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:SOURce
driver.trigger.afRf.generator.arb.set_source(source = '1')
```

Selects the trigger event source used to start or restart ARB file processing. To query a list of all supported sources, use method RsCma.Trigger.AfRf.Generator.Arb.Catalog.source.

param source
'Manual' Manual start via method RsCma.Trigger.AfRf.Generator.Arb.Manual.Execute.set
'Base1: External TRIG In' External trigger signal received at connector TRIG IN

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.generator.arb.clone()
```

Subgroups

6.19.1.1.1.1 Catalog

SCPI Command:

```
TRIGger:AFRF:GENerator<Instance>:ARB:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:CATalog:SOURce
value: List[str] = driver.trigger.afRf.generator.arb.catalog.get_source()
```

Returns a list of all trigger source values that can be selected via method RsCma.Trigger.AfRf.Generator.Arb.source.

return
trigger_sources: Comma-separated list of all strings, one string per source

6.19.1.1.2 Manual

class ManualCls

Manual commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.generator.arb.manual.clone()
```

Subgroups

6.19.1.1.3 Execute

SCPI Command:

```
TRIGger:AFRF:GENerator<Instance>:ARB:MANual:EXECute
```

class ExecuteCls

Execute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:MANual:EXECute
driver.trigger.afRf.generator.arb.manual.execute.set()
```

Generates a trigger event for the trigger source ‘Manual’.

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: TRIGger:AFRF:GENerator<Instance>:ARB:MANual:EXECute
driver.trigger.afRf.generator.arb.manual.execute.set_with_opc()
```

Generates a trigger event for the trigger source 'Manual'.

Same as set, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.19.1.2 Measurement

class MeasurementCls

Measurement commands group definition. 35 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.measurement.clone()
```

Subgroups

6.19.1.2.1 MultiEval

class MultiEvalCls

MultiEval commands group definition. 35 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.measurement.multiEval.clone()
```

Subgroups

6.19.1.2.1.1 Oscilloscope

class OscilloscopeCls

Oscilloscope commands group definition. 35 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.measurement.multiEval.oscilloscope.clone()
```

Subgroups

6.19.1.2.1.2 AudioInput

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:SOURce
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:COUPling
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:STATe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:ENABle
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:OFFSet
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:SLOPe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:MODE
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:THReshold
```

class AudioInputCls

AudioInput commands group definition. 8 total commands, 0 Subgroups, 8 group commands

get_coupling() → TriggerCouplingAin

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:COUPling
value: enums.TriggerCouplingAin = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.audioInput.get_coupling()
```

Couples the trigger settings for the AF input path to the trigger settings for another path.

return
trigger_coupling: NONE | DEMod | SIN | VOIP No coupling, coupling to RF path, to SPDIF path, to VoIP path

get_enable() → bool

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:ENABle
value: bool = driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.
↪get_enable()
```

No command help available

return
enable: No help available

get_mode() → TriggerMode

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:MODE
value: enums.TriggerMode = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.audioInput.get_mode()
```

Selects the repetition mode of the trigger system.

return
trigger_mode: SINGle | NORMal | AUTO | FRUN

get_offset() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:OFFSet
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪audioInput.get_offset()
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

return
trigger_offset: Range: 0 % to 100 %, Unit: %

get_slope() → SignalSlope

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:SLOPe
value: enums.SignalSlope = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.audioInput.get_slope()
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

return
trigger_slope: REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

get_source() → TriggerSourceAf

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:SOURce
value: enums.TriggerSourceAf = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.audioInput.get_source()
```

Selects a trigger event source for the AF input path.

return
trigger_source: AF1 | AF2 Connector AF1 IN or AF2 IN

get_state() → ArmedState

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:STATe
value: enums.ArmedState = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.audioInput.get_state()
```

Queries the state of the trigger system.

return
armed_state: OFF | ARMed | TRIGgered OFF The trigger system is disabled. The oscilloscope works as free-run measurement. ARMed The trigger system is armed and waits for a trigger event. TRIGgered A trigger event has occurred. The trigger system has not (yet) been rearmed.

get_threshold() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:AIN:THReshold
```

(continues on next page)

(continued from previous page)

```
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪audioInput.get_threshold()
```

Defines the trigger threshold for the AF input path.

return
threshold: Audio level threshold Range: -43 V to 43 V, Unit: V

set_coupling(*trigger_coupling: TriggerCouplingAin*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:COUPling
driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪coupling(trigger_coupling = enums.TriggerCouplingAin.DEMod)
```

Couples the trigger settings for the AF input path to the trigger settings for another path.

param trigger_coupling
NONE | DEMod | SIN | VOIP No coupling, coupling to RF path, to SPDIF path, to VoIP path

set_enable(*enable: bool*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:ENABle
driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪enable(enable = False)
```

No command help available

param enable
No help available

set_mode(*trigger_mode: TriggerMode*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:MODE
driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪mode(trigger_mode = enums.TriggerMode.AUTO)
```

Selects the repetition mode of the trigger system.

param trigger_mode
SINGle | NORMal | AUTO | FRUN

set_offset(*trigger_offset: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:OFFSet
driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪offset(trigger_offset = 1.0)
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

param trigger_offset
Range: 0 % to 100 %, Unit: %

set_slope(*trigger_slope: SignalSlope*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:SLOPe
driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪slope(trigger_slope = enums.SignalSlope.FEDGe)
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

param trigger_slope
REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

set_source(*trigger_source: TriggerSourceAf*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:AIN:SOURce
driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪source(trigger_source = enums.TriggerSourceAf.AF1)
```

Selects a trigger event source for the AF input path.

param trigger_source
AF1 | AF2 Connector AF1 IN or AF2 IN

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:AIN:THReshold
driver.trigger.afRf.measurement.multiEval.oscilloscope.audioInput.set_
↪threshold(threshold = 1.0)
```

Defines the trigger threshold for the AF input path.

param threshold
Audio level threshold Range: -43 V to 43 V, Unit: V

6.19.1.2.1.3 Demodulation

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:SOURce
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:COUPling
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:STATe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:ENABle
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:OFFSet
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:SLOPe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:MODE
```

class DemodulationCls

Demodulation commands group definition. 11 total commands, 4 Subgroups, 7 group commands

get_coupling() → TriggerCouplingDemod

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:COUPling
value: enums.TriggerCouplingDemod = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.demodulation.get_coupling()
```

Couples the trigger settings for the RF input path to the trigger settings for another path.

return
trigger_coupling: NONE | AIN | SIN | VOIP No coupling, coupling to AF path, to SPDIF path, to VoIP path

get_enable() → bool

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:ENABle
value: bool = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪demodulation.get_enable()
```

No command help available

return
enable: No help available

get_mode() → TriggerMode

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MODE
value: enums.TriggerMode = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.demodulation.get_mode()
```

Selects the repetition mode of the trigger system.

return
trigger_mode: SINGle | NORMal | AUTO | FRUN

get_offset() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:OFFSet
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪demodulation.get_offset()
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

return
trigger_offset: Range: 0 % to 100 %, Unit: %

get_slope() → SignalSlope

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:SLOPe
value: enums.SignalSlope = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.demodulation.get_slope()
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

return
trigger_slope: REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

get_source() → TriggerSourceDemod

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:SOURce
value: enums.TriggerSourceDemod = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.demodulation.get_source()
```

Selects a trigger event source for the RF input path.

return
trigger_source: LEFT | RIGHt | DEMod LEFT, RIGHt: left or right channel, for FM stereo only DEMod: demodulator output, not for FM stereo

get_state() → ArmedState

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:STATe
value: enums.ArmedState = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.demodulation.get_state()
```

Queries the state of the trigger system.

return
armed_state: OFF | ARMed | TRIGgered OFF The trigger system is disabled. The oscilloscope works as free-run measurement. ARMed The trigger system is armed and waits for a trigger event. TRIGgered A trigger event has occurred. The trigger system has not (yet) been rearmed.

set_coupling(*trigger_coupling: TriggerCouplingDemod*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:COUPling
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪coupling(trigger_coupling = enums.TriggerCouplingDemod.AIN)
```

Couples the trigger settings for the RF input path to the trigger settings for another path.

param trigger_coupling
NONE | AIN | SIN | VOIP No coupling, coupling to AF path, to SPDIF path, to VoIP path

set_enable(*enable: bool*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:ENABle
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪enable(enable = False)
```

No command help available

param enable
No help available

set_mode(*trigger_mode: TriggerMode*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MODE
```

(continues on next page)

(continued from previous page)

```
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪mode(trigger_mode = enums.TriggerMode.AUTO)
```

Selects the repetition mode of the trigger system.

param trigger_mode
SINGle | NORMal | AUTO | FRUN

set_offset(*trigger_offset: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:OFFSet
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪offset(trigger_offset = 1.0)
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

param trigger_offset
Range: 0 % to 100 %, Unit: %

set_slope(*trigger_slope: SignalSlope*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:SLOPe
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪slope(trigger_slope = enums.SignalSlope.FEDGe)
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

param trigger_slope
REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

set_source(*trigger_source: TriggerSourceDemod*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:SOURce
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.set_
↪source(trigger_source = enums.TriggerSourceDemod.DEMod)
```

Selects a trigger event source for the RF input path.

param trigger_source
LEFT | RIGHt | DEMod LEFT, RIGHt: left or right channel, for FM stereo only DEMod: demodulator output, not for FM stereo

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.clone()
```

Subgroups

6.19.1.2.1.4 Fdeviation

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:FDEViation:THReshold
```

class FdeviationCls

Fdeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_threshold() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:FDEViation:THReshold
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪demodulation.fdeviation.get_threshold()
```

Defines the trigger threshold for the RF input path, for FM and FM stereo demodulation.

return
threshold: Frequency deviation threshold Range: -96 kHz to 96 kHz, Unit: Hz

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:FDEViation:THReshold
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.fdeviation.
↪set_threshold(threshold = 1.0)
```

Defines the trigger threshold for the RF input path, for FM and FM stereo demodulation.

param threshold
Frequency deviation threshold Range: -96 kHz to 96 kHz, Unit: Hz

6.19.1.2.1.5 ModDepth

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:MDEPth:THReshold
```

class ModDepthCls

ModDepth commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_threshold() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MDEPth:THReshold
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪demodulation.modDepth.get_threshold()
```

Defines the trigger threshold for the RF input path, for AM demodulation.

return
threshold: Modulation depth threshold Range: -100 % to 100 %, Unit: %

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:MDEPth:THReshold
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.modDepth.
↪set_threshold(threshold = 1.0)
```

Defines the trigger threshold for the RF input path, for AM demodulation.

param threshold
Modulation depth threshold Range: -100 % to 100 %, Unit: %

6.19.1.2.1.6 Pdeviation

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:PDEViation:THReshold
```

class PdeviationCls

Pdeviation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_threshold() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:PDEViation:THReshold
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪demodulation.pdeviation.get_threshold()
```

Defines the trigger threshold for the RF input path, for PM demodulation.

return
threshold: Phase deviation threshold Range: -30 rad to 30 rad, Unit: rad

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:PDEViation:THReshold
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.pdeviation.
↪set_threshold(threshold = 1.0)
```

Defines the trigger threshold for the RF input path, for PM demodulation.

param threshold
Phase deviation threshold Range: -30 rad to 30 rad, Unit: rad

6.19.1.2.1.7 Ssb

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:DEModulation:SSB:THReshold
```

class SsbCls

Ssb commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_threshold() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:SSB:THReshold
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪demodulation.ssb.get_threshold()
```

Defines the trigger threshold for the RF input path, for SSB demodulation.

return
threshold: Audio level threshold Range: -30 V to 30 V, Unit: V

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:DEModulation:SSB:THReshold
driver.trigger.afRf.measurement.multiEval.oscilloscope.demodulation.ssb.set_
↪threshold(threshold = 1.0)
```

Defines the trigger threshold for the RF input path, for SSB demodulation.

param threshold
Audio level threshold Range: -30 V to 30 V, Unit: V

6.19.1.2.1.8 Spdif

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:SOURce
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:COUPling
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:STATe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:ENABle
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:OFFSet
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:SLOPe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:MODE
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:THReshold
```

class SpdifCls

Spdif commands group definition. 8 total commands, 0 Subgroups, 8 group commands

get_coupling() → TriggerCouplingDigital

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:COUPling
value: enums.TriggerCouplingDigital = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.spdif.get_coupling()
```

Couples the trigger settings for the SPDIF input path to the trigger settings for another path.

return
trigger_coupling: NONE | DEMod | AIN No coupling, coupling to RF path or to AF path

get_enable() → bool

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:ENABle
value: bool = driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.get_
↪enable()
```

No command help available

return
enable: No help available

get_mode() → TriggerMode

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:MODE
value: enums.TriggerMode = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.spdif.get_mode()
```

Selects the repetition mode of the trigger system.

return
trigger_mode: SINGle | NORMal | AUTO | FRUN

get_offset() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:OFFSet
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.get_
↪offset()
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

return
trigger_offset: Range: 0 % to 100 %, Unit: %

get_slope() → SignalSlope

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:SLOPe
value: enums.SignalSlope = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.spdif.get_slope()
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

return
trigger_slope: REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

get_source() → LeftRightDirection

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:SOURce
value: enums.LeftRightDirection = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.spdif.get_source()
```

Selects a trigger event source for the SPDIF input path.

return
trigger_source: LEFT | RIGHt Left or right SPDIF channel

get_state() → ArmedState

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:STATe
value: enums.ArmedState = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.spdif.get_state()
```

Queries the state of the trigger system.

return
armed_state: OFF | ARMed | TRIGgered OFF The trigger system is disabled. The oscilloscope works as free-run measurement. ARMed The trigger system is armed and waits for a trigger event. TRIGgered A trigger event has occurred. The trigger system has not (yet) been rearmed.

get_threshold() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SIN:THReshold
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.get_
↪threshold()
```

Defines the trigger threshold for the SPDIF input path.

return
threshold: Audio level threshold Range: -100 % to 100 %, Unit: %

set_coupling(*trigger_coupling: TriggerCouplingDigital*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:COUPling
driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.set_
↪coupling(trigger_coupling = enums.TriggerCouplingDigital.AIN)
```

Couples the trigger settings for the SPDIF input path to the trigger settings for another path.

param trigger_coupling
NONE | DEMod | AIN No coupling, coupling to RF path or to AF path

set_enable(*enable: bool*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:ENABle
driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.set_enable(enable␣
↪= False)
```

No command help available

param enable
No help available

set_mode(*trigger_mode: TriggerMode*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:MODE
driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.set_mode(trigger_
↪mode = enums.TriggerMode.AUTO)
```

Selects the repetition mode of the trigger system.

param trigger_mode
SINGle | NORMal | AUTO | FRUN

set_offset(*trigger_offset: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:OFFSet
driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.set_offset(trigger_
↪offset = 1.0)
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

param trigger_offset
Range: 0 % to 100 %, Unit: %

set_slope(*trigger_slope: SignalSlope*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:SLOPe
driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.set_slope(trigger_
↪slope = enums.SignalSlope.FEDGe)
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

param trigger_slope
REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

set_source(*trigger_source: LeftRightDirection*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:SIN:SOURce
driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.set_source(trigger_
↪source = enums.LeftRightDirection.LEFT)
```

Selects a trigger event source for the SPDIF input path.

param trigger_source
LEFT | RIGHt Left or right SPDIF channel

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:SIN:THReshold
driver.trigger.afRf.measurement.multiEval.oscilloscope.spdif.set_
↪threshold(threshold = 1.0)
```

Defines the trigger threshold for the SPDIF input path.

param threshold
Audio level threshold Range: -100 % to 100 %, Unit: %

6.19.1.2.1.9 Timeout

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:TOUT
```

class TimeoutCls

Timeout commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TimeoutStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables the timeout
- Timeout: float: Time interval during which a trigger event must occur Range: 0.2 s to 30 s, Unit: s

get() → TimeoutStruct

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:TOUT
value: TimeoutStruct = driver.trigger.afRf.measurement.multiEval.oscilloscope.
↪timeout.get()
```

Configures a timeout for the trigger modes 'Single' and 'Normal'.

return
structure: for return value, see the help for TimeoutStruct structure arguments.

set(*enable: bool*, *timeout: float = None*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:TOUT
driver.trigger.afRf.measurement.multiEval.oscilloscope.timeout.set(enable =␣
↪False, timeout = 1.0)
```

Configures a timeout for the trigger modes 'Single' and 'Normal'.

param enable
OFF | ON Enables the timeout

param timeout
Time interval during which a trigger event must occur Range: 0.2 s to 30 s, Unit: s

6.19.1.2.1.10 Voip

SCPI Command:

```
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:ENABle
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:THReshold
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:MODE
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:SLOPe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:OFFSet
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:STATe
TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:COUPling
```

class VoipCls

Voip commands group definition. 7 total commands, 0 Subgroups, 7 group commands

get_coupling() → TriggerCouplingDigital

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:COUPling
value: enums.TriggerCouplingDigital = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.voip.get_coupling()
```

Couples the trigger settings for the VoIP input path to the trigger settings for another path.

return
trigger_coupling: NONE | DEMod | AIN No coupling, coupling to RF path or to AF path

get_enable() → bool

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:ENABle
value: bool = driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.get_
↪enable()
```

No command help available

return
enable: No help available

get_mode() → TriggerMode

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:MODE
value: enums.TriggerMode = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.voip.get_mode()
```

Selects the repetition mode of the trigger system.

return
trigger_mode: SINGle | NORMal | AUTO | FRUN

get_offset() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:OFFSet
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.get_
↪offset()
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

return
trigger_offset: Range: 0 % to 100 %, Unit: %

get_slope() → SignalSlope

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:SLOPe
value: enums.SignalSlope = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.voip.get_slope()
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

return
trigger_slope: REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

get_state() → ArmedState

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:STATe
value: enums.ArmedState = driver.trigger.afRf.measurement.multiEval.
↪oscilloscope.voip.get_state()
```

Queries the state of the trigger system.

return
armed_state: OFF | ARMed | TRIGgered OFF The trigger system is disabled. The oscilloscope works as free-run measurement. ARMed The trigger system is armed and waits for a trigger event. TRIGgered A trigger event has occurred. The trigger system has not (yet) been rearmed.

get_threshold() → float

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:THReshold
value: float = driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.get_
↪threshold()
```

Defines the trigger threshold for the VoIP input path.

return
threshold: Audio level threshold Range: -100 % to 100 %, Unit: %

set_coupling(*trigger_coupling: TriggerCouplingDigital*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:COUPling
driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.set_
↪coupling(trigger_coupling = enums.TriggerCouplingDigital.AIN)
```

Couples the trigger settings for the VoIP input path to the trigger settings for another path.

param trigger_coupling
NONE | DEMod | AIN No coupling, coupling to RF path or to AF path

set_enable(*enable: bool*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:ENABle
driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.set_enable(enable =␣
↪False)
```

No command help available

param enable
No help available

set_mode(*trigger_mode: TriggerMode*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:MODE
driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.set_mode(trigger_
↪mode = enums.TriggerMode.AUTO)
```

Selects the repetition mode of the trigger system.

param trigger_mode
SINGle | NORMal | AUTO | FRUN

set_offset(*trigger_offset: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:OFFSet
driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.set_offset(trigger_
↪offset = 1.0)
```

Defines a trigger offset, shifting the measured trace relative to the trigger event, so that the trace starts earlier. The offset is specified as a percentage of the measurement time for a single trace.

param trigger_offset
Range: 0 % to 100 %, Unit: %

set_slope(*trigger_slope: SignalSlope*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>:MEValuation:OSCilloscope:VOIP:SLOPe
driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.set_slope(trigger_
↪slope = enums.SignalSlope.FEDGe)
```

Selects whether the trigger event is generated by signals rising through the threshold or falling through the threshold.

param trigger_slope
REDGe | FEDGe REDGe Rising signal (rising edge) FEDGe Falling signal (falling edge)

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:AFRF:MEASurement<Instance>
↪:MEValuation:OSCilloscope:VOIP:THReshold
driver.trigger.afRf.measurement.multiEval.oscilloscope.voip.set_
↪threshold(threshold = 1.0)
```

Defines the trigger threshold for the VoIP input path.

param threshold
Audio level threshold Range: -100 % to 100 %, Unit: %

6.19.2 Base

class BaseCls

Base 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.base.clone()
```

Subgroups

6.19.2.1 Out

SCPI Command:

```
TRIGger:BASE:OUT:SOURce
```

class OutCls

Out commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_source() → str

```
# SCPI: TRIGger:BASE:OUT:SOURce
value: str = driver.trigger.base.out.get_source()
```

Selects the output trigger signal to be routed to the TRIG OUT connector.

return
source: Source as string, examples: 'No Connection' TRIG OUT connector deactivated 'Base1: External TRIG In' Trigger signal from TRIG IN connector 'AFRF Gen1: ...' Trigger signal from processed waveform file

set_source(*source: str*) → None

```
# SCPI: TRIGger:BASE:OUT:SOURce
driver.trigger.base.out.set_source(source = '1')
```

Selects the output trigger signal to be routed to the TRIG OUT connector.

param source
Source as string, examples: 'No Connection' TRIG OUT connector deactivated 'Base1: External TRIG In' Trigger signal from TRIG IN connector 'AFRF Gen1: ...' Trigger signal from processed waveform file

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.base.out.clone()
```

Subgroups

6.19.2.1.1 Catalog

SCPI Command:

```
TRIGger:BASE:OUT:CATalog:SOURce
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_source() → List[str]

```
# SCPI: TRIGger:BASE:OUT:CATalog:SOURce
value: List[str] = driver.trigger.base.out.catalog.get_source()
```

Lists all trigger source values that can be selected via method RsCma.Trigger.Base.Out.source.

return
source_list: Comma-separated list of all supported values Each value is represented as a string.

6.19.3 GprfMeasurement

class GprfMeasurementCls

GprfMeasurement commands group definition. 31 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprfMeasurement.clone()
```

Subgroups

6.19.3.1 FftSpecAn

SCPI Command:

```
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
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.gprfMeasurement.fftSpecAn.get_mgap()
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for the trigger source 'IF Power'.

return
minimum_gap: Range: 0 s to 0.01 s, Unit: s

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OFFSet
value: float = driver.trigger.gprfMeasurement.fftSpecAn.get_offset()
```

Defines a trigger offset for the FIXed trigger offset mode, see method RsCma.Trigger.GprfMeasurement.FftSpecAn.omode.

return
offset: Range: -0.15 s to 0.15 s, Unit: s

get_omode() → FftOffsetMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
value: enums.FftOffsetMode = driver.trigger.gprfMeasurement.fftSpecAn.get_
↪omode()
```

Selects the trigger offset mode.

return
offset_mode: VARiable | FIXed VARiable Variable trigger offset within a configurable range FIXed Static configurable trigger offset

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprfMeasurement.fftSpecAn.get_
↪slope()
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for trigger source 'IF Power'.

return
event: REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
value: str = driver.trigger.gprfMeasurement.fftSpecAn.get_source()
```

Selects a trigger event source for FFT spectrum analysis. To query a list of all supported sources, use method RsCma. Trigger.GprfMeasurement.FftSpecAn.Catalog.source.

return
source: Source as string, examples: 'Free Run' Immediate start without trigger signal 'IF Power' Trigger by IF power steps 'Base1: External TRIG In' Trigger signal at connector TRIG IN 'AFRF Gen1: ...' Trigger by processed waveform file

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:THReshold
value: float = driver.trigger.gprfMeasurement.fftSpecAn.get_threshold()
```

Defines the trigger threshold for trigger source 'IF Power'.

return
threshold: Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
value: float or bool = driver.trigger.gprfMeasurement.fftSpecAn.get_timeout()
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

return
timeout: (float or boolean) Range: 0.01 s to 300 s, Unit: s

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:MGAP
driver.trigger.gprfMeasurement.fftSpecAn.set_mgap(minimum_gap = 1.0)
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for the trigger source 'IF Power'.

param minimum_gap
Range: 0 s to 0.01 s, Unit: s

set_offset(*offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OFFSet
driver.trigger.gprfMeasurement.fftSpecAn.set_offset(offset = 1.0)
```

Defines a trigger offset for the FIXed trigger offset mode, see method RsCma.Trigger.GprfMeasurement.FftSpecAn.omode.

param offset
Range: -0.15 s to 0.15 s, Unit: s

set_omode(*offset_mode: FftOffsetMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OMODe
driver.trigger.gprfMeasurement.fftSpecAn.set_omode(offset_mode = enums.
↪FftOffsetMode.FIXed)
```

Selects the trigger offset mode.

param offset_mode
VARiable | FIXed VARiable Variable trigger offset within a configurable range FIXed Static configurable trigger offset

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SLOPe
driver.trigger.gprfMeasurement.fftSpecAn.set_slope(event = enums.SignalSlopeExt.
↪FALLing)
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for trigger source 'IF Power'.

param event
REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:SOURce
driver.trigger.gprfMeasurement.fftSpecAn.set_source(source = '1')
```

Selects a trigger event source for FFT spectrum analysis. To query a list of all supported sources, use method RsCma. Trigger.GprfMeasurement.FftSpecAn.Catalog.source.

param source
Source as string, examples: 'Free Run' Immediate start without trigger signal 'IF Power' Trigger by IF power steps 'Base1: External TRIG In' Trigger signal at connector TRIG IN 'AFRF Gen1: ...' Trigger by processed waveform file

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:THReshold
driver.trigger.gprfMeasurement.fftSpecAn.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for trigger source 'IF Power'.

param threshold
Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:TOUT
driver.trigger.gprfMeasurement.fftSpecAn.set_timeout(timeout = 1.0)
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

param timeout
(float or boolean) Range: 0.01 s to 300 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprfMeasurement.fftSpecAn.clone()
```

Subgroups

6.19.3.1.1 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.gprfMeasurement.fftSpecAn.catalog.get_source()
```

Returns a list of all trigger source values that can be selected via method RsCma.Trigger.GprfMeasurement.FftSpecAn. source.

return
trigger_sources: Comma-separated list of all strings, one string per source

6.19.3.1.2 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: Range: -0.15 s to OffsetStop, Unit: s
- Offset_Stop: float: Range: OffsetStart to 0.15 s, Unit: s

get() → OsStopStruct

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:FFTSanalyzer:OSSTop
value: OsStopStruct = driver.trigger.gprfMeasurement.fftSpecAn.osStop.get()
```

Defines the initial and final trigger offset for the VARiable trigger offset mode, see method RsCma.Trigger.GprfMeasurement.FftSpecAn.omode.

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.gprfMeasurement.fftSpecAn.osStop.set(offset_start = 1.0, offset_
↪stop = 1.0)
```

Defines the initial and final trigger offset for the VARiable trigger offset mode, see method RsCma.Trigger.GprfMeasurement.FftSpecAn.omode.

param offset_start
Range: -0.15 s to OffsetStop, Unit: s

param offset_stop
Range: OffsetStart to 0.15 s, Unit: s

6.19.3.2 IqRecorder

SCPI Command:

```
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:MGAP
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SLOPe
```

class IqRecorderCls

IqRecorder commands group definition. 7 total commands, 1 Subgroups, 6 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:MGAP
value: float = driver.trigger.gprfMeasurement.iqRecorder.get_mgap()
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for trigger source 'IF Power'.

return
minimum_gap: Range: 0 s to 0.01 s, Unit: s

get_offset() → int

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
value: int = driver.trigger.gprfMeasurement.iqRecorder.get_offset()
```

Delays the start of the measurement relative to the trigger event. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

return
trigger_offset: Range: 0 samples to 64E+6 samples

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprfMeasurement.iqRecorder.get_
↪slope()
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for the trigger source 'IF Power'.

return
event: REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
value: str = driver.trigger.gprfMeasurement.iqRecorder.get_source()
```

Selects a trigger event source for I/Q recorder measurements. To query a list of all supported sources, use method RsCma. Trigger.GprfMeasurement.IqRecorder.Catalog.source.

return
source: Source as string, examples: ‘Free Run’ Immediate start without trigger signal ‘IF Power’ Trigger by IF power steps ‘Base1: External TRIG In’ Trigger signal at connector TRIG IN ‘AFRF Gen1: …’ Trigger by processed waveform file

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
value: float = driver.trigger.gprfMeasurement.iqRecorder.get_threshold()
```

Defines the trigger threshold for trigger source ‘IF Power’.

return
threshold: Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
value: float or bool = driver.trigger.gprfMeasurement.iqRecorder.get_timeout()
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source ‘IF Power’ and for trigger signals at TRIG IN.

return
timeout: (float or boolean) Range: 0.01 s to 300 s, Unit: s

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:MGAP
driver.trigger.gprfMeasurement.iqRecorder.set_mgap(minimum_gap = 1.0)
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for trigger source ‘IF Power’.

param minimum_gap
Range: 0 s to 0.01 s, Unit: s

set_offset(*trigger_offset: int*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:OFFSet
driver.trigger.gprfMeasurement.iqRecorder.set_offset(trigger_offset = 1)
```

Delays the start of the measurement relative to the trigger event. This setting is relevant for the trigger source ‘IF Power’ and for trigger signals at TRIG IN.

param trigger_offset
Range: 0 samples to 64E+6 samples

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SLOPe
driver.trigger.gprfMeasurement.iqRecorder.set_slope(event = enums.
↪SignalSlopeExt.FALLing)
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for the trigger source ‘IF Power’.

param event
REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:SOURce
driver.trigger.gprfMeasurement.iqRecorder.set_source(source = '1')
```

Selects a trigger event source for I/Q recorder measurements. To query a list of all supported sources, use method RsCma. Trigger.GprfMeasurement.IqRecorder.Catalog.source.

param source
Source as string, examples: 'Free Run' Immediate start without trigger signal 'IF Power' Trigger by IF power steps 'Base1: External TRIG In' Trigger signal at connector TRIG IN 'AFRF Gen1: ...' Trigger by processed waveform file

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:THReshold
driver.trigger.gprfMeasurement.iqRecorder.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for trigger source 'IF Power'.

param threshold
Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:IQRecorder:TOUT
driver.trigger.gprfMeasurement.iqRecorder.set_timeout(timeout = 1.0)
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

param timeout
(float or boolean) Range: 0.01 s to 300 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprfMeasurement.iqRecorder.clone()
```

Subgroups

6.19.3.2.1 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.gprfMeasurement.iqRecorder.catalog.get_
↪source()
```

Returns a list of all trigger source values that can be selected via method RsCma.Trigger.GprfMeasurement.IqRecorder. source.

return
trigger_sources: Comma-separated list of all strings, one string per source

6.19.3.3 Power

SCPI Command:

```
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. 8 total commands, 1 Subgroups, 7 group commands

get_mgap() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MGAP
value: float = driver.trigger.gprfMeasurement.power.get_mgap()
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for trigger source 'IF Power'.

return
minimum_gap: Range: 0 s to 0.01 s, Unit: s

get_mode() → PowerMode

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MODE
value: enums.PowerMode = driver.trigger.gprfMeasurement.power.get_mode()
```

Selects the trigger mode for the measurement. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

return
mode: ONCE | SWEep | ALL ONCE Only the start of the measurement is triggered SWEep Each measurement cycle is triggered ALL Each measurement interval is triggered

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:OFFSet
value: float = driver.trigger.gprfMeasurement.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: Range: 0 s to 1 s, Unit: s

get_slope() → SignalSlopeExt

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SLOPe
value: enums.SignalSlopeExt = driver.trigger.gprfMeasurement.power.get_slope()
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for the trigger source 'IF Power'.

return
event: REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
value: str = driver.trigger.gprfMeasurement.power.get_source()
```

Selects a trigger event source for power measurements. To query a list of all supported sources, use method RsCma.Trigger. GprfMeasurement.Power.Catalog.source.

return
source: Source as string, examples: 'Free Run' Immediate start without trigger signal 'IF Power' Trigger by IF power steps 'Base1: External TRIG In' Trigger signal at connector TRIG IN 'AFRF Gen1: ...' Trigger by processed waveform file

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:THReshold
value: float = driver.trigger.gprfMeasurement.power.get_threshold()
```

Defines the trigger threshold for trigger source 'IF Power'.

return
threshold: Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:TOUT
value: float or bool = driver.trigger.gprfMeasurement.power.get_timeout()
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

return
timeout: (float or boolean) Range: 0.01 s to 300 s, Unit: s

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MGAP
driver.trigger.gprfMeasurement.power.set_mgap(minimum_gap = 1.0)
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for trigger source 'IF Power'.

param minimum_gap
Range: 0 s to 0.01 s, Unit: s

set_mode(*mode: PowerMode*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:MODE
driver.trigger.gprfMeasurement.power.set_mode(mode = enums.PowerMode.ALL)
```

Selects the trigger mode for the measurement. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

param mode
ONCE | SWEep | ALL ONCE Only the start of the measurement is triggered SWEep Each measurement cycle is triggered ALL Each measurement interval is triggered

set_offset(*offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:OFFSet
driver.trigger.gprfMeasurement.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
Range: 0 s to 1 s, Unit: s

set_slope(*event: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SLOPe
driver.trigger.gprfMeasurement.power.set_slope(event = enums.SignalSlopeExt.
↪FALLing)
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for the trigger source 'IF Power'.

param event
REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:SOURce
driver.trigger.gprfMeasurement.power.set_source(source = '1')
```

Selects a trigger event source for power measurements. To query a list of all supported sources, use method RsCma.Trigger. GprfMeasurement.Power.Catalog.source.

param source
Source as string, examples: 'Free Run' Immediate start without trigger signal 'IF Power' Trigger by IF power steps 'Base1: External TRIG In' Trigger signal at connector TRIG IN 'AFRF Gen1: ...' Trigger by processed waveform file

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:THReshold
driver.trigger.gprfMeasurement.power.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for trigger source 'IF Power'.

param threshold
Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

set_timeout(*timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:POWer:TOUT
driver.trigger.gprfMeasurement.power.set_timeout(timeout = 1.0)
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source 'IF Power' and for trigger signals at TRIG IN.

param timeout
(float or boolean) Range: 0.01 s to 300 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprfMeasurement.power.clone()
```

Subgroups

6.19.3.3.1 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.gprfMeasurement.power.catalog.get_source()
```

Returns a list of all trigger source values that can be selected via method RsCma.Trigger.GprfMeasurement.Power.source.

return
trigger_sources: Comma-separated list of all strings, one string per source

6.19.3.4 Spectrum

SCPI Command:

```
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SOURce
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.gprfMeasurement.spectrum.get_mgap()
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for trigger source 'Video'.

return
minimum_gap: Range: 0 s to 0.01 s, Unit: s

get_offset() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:OFFSet
value: float = driver.trigger.gprfMeasurement.spectrum.get_offset()
```

Defines a delay time for triggered zero span measurements. The trigger offset delays the start of the measurement relative to the trigger event. This command is relevant for the trigger source 'Video' and for trigger signals at TRIG IN.

return
trigger_offset: 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.gprfMeasurement.spectrum.get_
↪slope()
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for the trigger source 'Video'.

return
slope: REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

get_source() → str

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SOURce
value: str = driver.trigger.gprfMeasurement.spectrum.get_source()
```

Selects a trigger event source for spectrum analysis. To query a list of all supported sources, use method RsCma.Trigger. GprfMeasurement.Spectrum.Catalog.source.

return
source: Source as string, examples: 'Free Run' Immediate start without trigger signal 'Video' Trigger by video power steps 'Base1: External TRIG In' Trigger signal at connector TRIG IN 'AFRF Gen1: ...' Trigger by processed waveform file

get_threshold() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
value: float = driver.trigger.gprfMeasurement.spectrum.get_threshold()
```

Defines the trigger threshold for trigger source 'Video'.

return
threshold: Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

get_timeout() → float

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:TOUT
value: float = driver.trigger.gprfMeasurement.spectrum.get_timeout()
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source 'Video' and for trigger signals at TRIG IN.

return
trigger_timeout: Range: 0.01 s to 300 s, Unit: s

set_mgap(*minimum_gap: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:MGAP
driver.trigger.gprfMeasurement.spectrum.set_mgap(minimum_gap = 1.0)
```

Defines the minimum duration of the power-down periods (gaps) between two triggered power pulses. This setting is relevant for trigger source 'Video'.

param minimum_gap
Range: 0 s to 0.01 s, Unit: s

set_offset(*trigger_offset: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:OFFSet
driver.trigger.gprfMeasurement.spectrum.set_offset(trigger_offset = 1.0)
```

Defines a delay time for triggered zero span measurements. The trigger offset delays the start of the measurement relative to the trigger event. This command is relevant for the trigger source 'Video' and for trigger signals at TRIG IN.

param trigger_offset
Range: -0.5 s to 0.5 s, Unit: s

set_slope(*slope: SignalSlopeExt*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SLOPe
driver.trigger.gprfMeasurement.spectrum.set_slope(slope = enums.SignalSlopeExt.
↪FALLing)
```

Selects whether the trigger event is generated at the rising or at the falling edge of the trigger pulse. This command is relevant for the trigger source 'Video'.

param slope
REDGe | FEDGe | RISing | FALLing REDGe, RISing Rising edge FEDGe, FALLing Falling edge

set_source(*source: str*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:SOURce
driver.trigger.gprfMeasurement.spectrum.set_source(source = '1')
```

Selects a trigger event source for spectrum analysis. To query a list of all supported sources, use method RsCma.Trigger. GprfMeasurement.Spectrum.Catalog.source.

param source
Source as string, examples: 'Free Run' Immediate start without trigger signal 'Video' Trigger by video power steps 'Base1: External TRIG In' Trigger signal at connector TRIG IN 'AFRF Gen1: ...' Trigger by processed waveform file

set_threshold(*threshold: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:THReshold
driver.trigger.gprfMeasurement.spectrum.set_threshold(threshold = 1.0)
```

Defines the trigger threshold for trigger source 'Video'.

param threshold
Range: -50 dB to 0 dB, Unit: dB (full scale, i.e. relative to expected power minus external attenuation)

set_timeout(*trigger_timeout: float*) → None

```
# SCPI: TRIGger:GPRF:MEASurement<Instance>:SPECtrum:TOUT
driver.trigger.gprfMeasurement.spectrum.set_timeout(trigger_timeout = 1.0)
```

Specifies the time after which an initiated measurement must have received a trigger event. If no trigger event is received, the measurement is stopped in remote control mode. In manual operation mode, a trigger timeout is indicated. This setting is relevant for the trigger source 'Video' and for trigger signals at TRIG IN.

param trigger_timeout
Range: 0.01 s to 300 s, Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.trigger.gprfMeasurement.spectrum.clone()
```

Subgroups

6.19.3.4.1 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.gprfMeasurement.spectrum.catalog.get_source()
```

Returns a list of all trigger source values that can be selected via method RsCma.Trigger.GprfMeasurement.Spectrum.source.

return

trigger_sources: Comma-separated list of all strings, one string per source

6.20 Unit

SCPI Command:

```
UNIT:FSCale
UNIT:CONDuctance
UNIT:CHARge
UNIT:CAPacity
UNIT:ENERgy
UNIT:FREQuency
UNIT:RESistor
UNIT:VOLTage
UNIT:ANGLe
UNIT:LENGth
UNIT:CURRent
UNIT:POWer
UNIT:TEMPerature
UNIT:TIME
```

class UnitCls

Unit commands group definition. 14 total commands, 0 Subgroups, 14 group commands

get_angle() → DefaultUnitAngle

```
# SCPI: UNIT:ANGLe
value: enums.DefaultUnitAngle = driver.unit.get_angle()
```

Defines the default unit for angle settings and angle results. This command affects remote control commands. It does not affect the angle value presentation at the GUI.

return
default_unit_angle: No help available

get_capacity() → DefaultUnitCapacity

```
# SCPI: UNIT:CAPacity
value: enums.DefaultUnitCapacity = driver.unit.get_capacity()
```

No command help available

return
default_unit_capacity: No help available

get_charge() → DefaultUnitCharge

```
# SCPI: UNIT:CHARge
value: enums.DefaultUnitCharge = driver.unit.get_charge()
```

No command help available

return
default_unit_charge: No help available

get_conductance() → DefaultUnitConductance

```
# SCPI: UNIT:CONDuctance
value: enums.DefaultUnitConductance = driver.unit.get_conductance()
```

No command help available

return
default_unit_conductance: No help available

get_current() → DefaultUnitCurrent

```
# SCPI: UNIT:CURRent
value: enums.DefaultUnitCurrent = driver.unit.get_current()
```

No command help available

return
default_unit_current: No help available

get_energy() → DefaultUnitEnergy

```
# SCPI: UNIT:ENERgy
value: enums.DefaultUnitEnergy = driver.unit.get_energy()
```

No command help available

return
default_unit_energy: No help available

get_frequency() → DefaultUnitFrequency

```
# SCPI: UNIT:FREQuency
value: enums.DefaultUnitFrequency = driver.unit.get_frequency()
```

Defines the default unit for frequency settings and frequency results. This command affects remote control commands. It does not affect the frequency value presentation at the GUI.

return
default_unit_frequency: No help available

get_fscale() → DefaultUnitFullScale

```
# SCPI: UNIT:FSCale
value: enums.DefaultUnitFullScale = driver.unit.get_fscale()
```

Defines the default unit for full-scale settings and full-scale results. This command affects remote control commands. It does not affect the presentation at the GUI.

return
default_unit_full_scale: No help available

get_length() → DefaultUnitLenght

```
# SCPI: UNIT:LENGth
value: enums.DefaultUnitLenght = driver.unit.get_length()
```

No command help available

return
default_unit_lenght: No help available

get_power() → DefaultUnitPower

```
# SCPI: UNIT:POWer
value: enums.DefaultUnitPower = driver.unit.get_power()
```

Defines the default unit for power settings and power results. This command affects remote control commands. It does not affect the power value presentation at the GUI.

return
default_unit_power: No help available

get_resistor() → DefaultUnitResistor

```
# SCPI: UNIT:RESistor
value: enums.DefaultUnitResistor = driver.unit.get_resistor()
```

No command help available

return
default_unit_resistor: No help available

get_temperature() → DefaultUnitTemperature

```
# SCPI: UNIT:TEMPerature
value: enums.DefaultUnitTemperature = driver.unit.get_temperature()
```

No command help available

return
default_unit_temperature: No help available

get_time() → DefaultUnitTime

```
# SCPI: UNIT:TIME
value: enums.DefaultUnitTime = driver.unit.get_time()
```

Defines the default unit for time settings and time results. This command affects remote control commands. It does not affect the time value presentation at the GUI.

return
default_unit_time: No help available

get_voltage() → DefaultUnitVoltage

```
# SCPI: UNIT:VOLTage
value: enums.DefaultUnitVoltage = driver.unit.get_voltage()
```

Defines the default unit for voltage settings and voltage results. This command affects remote control commands. It does not affect the voltage value presentation at the GUI.

return
default_unit_voltage: No help available

set_angle(*default_unit_angle: DefaultUnitAngle*) → None

```
# SCPI: UNIT:ANGLe
driver.unit.set_angle(default_unit_angle = enums.DefaultUnitAngle.DEG)
```

Defines the default unit for angle settings and angle results. This command affects remote control commands. It does not affect the angle value presentation at the GUI.

param default_unit_angle
DEG | RAD Means: degrees, radians

set_capacity(*default_unit_capacity: DefaultUnitCapacity*) → None

```
# SCPI: UNIT:CAPacity
driver.unit.set_capacity(default_unit_capacity = enums.DefaultUnitCapacity.AF)
```

No command help available

param default_unit_capacity
No help available

set_charge(*default_unit_charge: DefaultUnitCharge*) → None

```
# SCPI: UNIT:CHARge
driver.unit.set_charge(default_unit_charge = enums.DefaultUnitCharge.AC)
```

No command help available

param default_unit_charge
No help available

set_conductance(*default_unit_conductance: DefaultUnitConductance*) → None

```
# SCPI: UNIT:CONDuctance
driver.unit.set_conductance(default_unit_conductance = enums.
↪DefaultUnitConductance.ASIE)
```

No command help available

param default_unit_conductance
No help available

set_current(*default_unit_current: DefaultUnitCurrent*) → None

```
# SCPI: UNIT:CURRent
driver.unit.set_current(default_unit_current = enums.DefaultUnitCurrent.A)
```

No command help available

param default_unit_current
No help available

set_energy(*default_unit_energy: DefaultUnitEnergy*) → None

```
# SCPI: UNIT:ENERgy
driver.unit.set_energy(default_unit_energy = enums.DefaultUnitEnergy.AJ)
```

No command help available

param default_unit_energy
No help available

set_frequency(*default_unit_frequency: DefaultUnitFrequency*) → None

```
# SCPI: UNIT:FREQuency
driver.unit.set_frequency(default_unit_frequency = enums.DefaultUnitFrequency.
↪AHZ)
```

Defines the default unit for frequency settings and frequency results. This command affects remote control commands. It does not affect the frequency value presentation at the GUI.

param default_unit_frequency
HZ | KHZ | MHZ | GHZ Means: Hz, kHz, MHz, GHz

set_fscale(*default_unit_full_scale: DefaultUnitFullScale*) → None

```
# SCPI: UNIT:FSCale
driver.unit.set_fscale(default_unit_full_scale = enums.DefaultUnitFullScale.
↪DBFS)
```

Defines the default unit for full-scale settings and full-scale results. This command affects remote control commands. It does not affect the presentation at the GUI.

param default_unit_full_scale
FS | PCT Means: FS, %

set_length(*default_unit_lenght: DefaultUnitLenght*) → None

```
# SCPI: UNIT:LENGth
driver.unit.set_length(default_unit_lenght = enums.DefaultUnitLenght.AM)
```

No command help available

param default_unit_lenght
No help available

set_power(*default_unit_power: DefaultUnitPower*) → None

```
# SCPI: UNIT:POWer
driver.unit.set_power(default_unit_power = enums.DefaultUnitPower.AW)
```

Defines the default unit for power settings and power results. This command affects remote control commands. It does not affect the power value presentation at the GUI.

param default_unit_power
W | MIW | DBMW | DBW Means: W, mW, dBm, dBW

set_resistor(*default_unit_resistor: DefaultUnitResistor*) → None

```
# SCPI: UNIT:RESistor
driver.unit.set_resistor(default_unit_resistor = enums.DefaultUnitResistor.AOHM)
```

No command help available

param default_unit_resistor
No help available

set_temperature(*default_unit_temperature: DefaultUnitTemperature*) → None

```
# SCPI: UNIT:TEMPerature
driver.unit.set_temperature(default_unit_temperature = enums.
↪DefaultUnitTemperature.C)
```

No command help available

param default_unit_temperature
No help available

set_time(*default_unit_time: DefaultUnitTime*) → None

```
# SCPI: UNIT:TIME
driver.unit.set_time(default_unit_time = enums.DefaultUnitTime.AS)
```

Defines the default unit for time settings and time results. This command affects remote control commands. It does not affect the time value presentation at the GUI.

param default_unit_time
M | S | MS | US | NS Means: minutes, seconds, ms, us, ns

set_voltage(*default_unit_voltage: DefaultUnitVoltage*) → None

```
# SCPI: UNIT:VOLTage
driver.unit.set_voltage(default_unit_voltage = enums.DefaultUnitVoltage.AV)
```

Defines the default unit for voltage settings and voltage results. This command affects remote control commands. It does not affect the voltage value presentation at the GUI.

param default_unit_voltage
V | MV | UV | DBV | DBMV | DBUV Means: V, mV, uV, dBV, dBmV, dBuV

6.21 Vse

class VseCls

Vse commands group definition. 87 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.clone()
```

Subgroups

6.21.1 Measurement

SCPI Command:

```
STOP:VSE:MEASurement<Instance>
ABORt:VSE:MEASurement<Instance>
```

class MeasurementCls

Measurement commands group definition. 87 total commands, 19 Subgroups, 2 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:VSE:MEASurement<Instance>
driver.vse.measurement.abort()
```

Stops the measurement.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

stop() → None

```
# SCPI: STOP:VSE:MEASurement<Instance>
driver.vse.measurement.stop()
```

Pauses the measurement.

stop_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: STOP:VSE:MEASurement<Instance>
driver.vse.measurement.stop_with_opc()
```

Pauses the measurement.

Same as stop, but waits for the operation to complete before continuing further. Use the RsCma.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.clone()
```

Subgroups

6.21.1.1 Cons

class ConsCls

Cons 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.vse.measurement.cons.clone()
```

Subgroups

6.21.1.1.1 Frequency

class FrequencyCls

Frequency 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.vse.measurement.cons.frequency.clone()
```

Subgroups

6.21.1.1.1.1 Current

SCPI Command:

```
READ:VSE:MEASurement<Instance>:CONS:FREQuency:CURRent
FETCh:VSE:MEASurement<Instance>:CONS:FREQuency:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:VSE:MEASurement<Instance>:CONS:FREQuency:CURRent
value: List[float] = driver.vse.measurement.cons.frequency.current.fetch()
```

Query the list of constellation frequencies.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
constellation: List of frequency deviations over the carrier frequency. Unit: Hz

read() → List[float]

```
# SCPI: READ:VSE:MEASurement<Instance>:CONS:FREQuency:CURRent
value: List[float] = driver.vse.measurement.cons.frequency.current.read()
```

Query the list of constellation frequencies.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
constellation: List of frequency deviations over the carrier frequency. Unit: Hz

6.21.1.1.2 Iq

class IqCls

Iq 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.vse.measurement.cons.iq.clone()
```

Subgroups

6.21.1.1.2.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:CONS:IQ:CURRent
READ:VSE:MEASurement<Instance>:CONS:IQ:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Xval: List[float]: List of the in-phase values.
- Yval: List[float]: List of the quadrature values.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:CONS:IQ:CURRent
value: ResultData = driver.vse.measurement.cons.iq.current.fetch()
```

Query the IQ constellation values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:CONS:IQ:CURRent
value: ResultData = driver.vse.measurement.cons.iq.current.read()
```

Query the IQ constellation values.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.2 Dmr

class DmrCls

Dmr commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.dmr.clone()
```

Subgroups

6.21.1.2.1 Symbols

class SymbolsCls

Symbols 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.vse.measurement.dmr.symbols.clone()
```

Subgroups

6.21.1.2.1.1 Binary

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:DMR:SYMBols:BINary
READ:VSE:MEASurement<Instance>:DMR:SYMBols:BINary
```

class BinaryCls

Binary commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of binary values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:DMR:SYMBols:BINary
value: ResultData = driver.vse.measurement.dmr.symbols.binary.fetch()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:DMR:SYMBols:BINary
value: ResultData = driver.vse.measurement.dmr.symbols.binary.read()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.2.1.2 Hexadecimal

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:DMR:SYMBols:HEXadecimal
READ:VSE:MEASurement<Instance>:DMR:SYMBols:HEXadecimal
```

class HexadecimalCls

Hexadecimal commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of hexadecimal values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:DMR:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.dmr.symbols.hexadecimal.fetch()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:DMR:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.dmr.symbols.hexadecimal.read()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.3 Dpmr

class DpmrCls

Dpmr commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.dpmr.clone()
```

Subgroups

6.21.1.3.1 Symbols

class SymbolsCls

Symbols 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.vse.measurement.dpmr.symbols.clone()
```

Subgroups

6.21.1.3.1.1 Binary

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:DPMR:SYMBols:BINary
READ:VSE:MEASurement<Instance>:DPMR:SYMBols:BINary
```

class BinaryCls

Binary commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'

- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of binary values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:DPMR:SYMBols:BINary
value: ResultData = driver.vse.measurement.dpmr.symbols.binary.fetch()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:DPMR:SYMBols:BINary
value: ResultData = driver.vse.measurement.dpmr.symbols.binary.read()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.3.1.2 Hexadecimal

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:DPMR:SYMBols:HEXadecimal
READ:VSE:MEASurement<Instance>:DPMR:SYMBols:HEXadecimal
```

class HexadecimalCls

Hexadecimal commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of hexadecimal values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:DPMR:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.dpmr.symbols.hexadecimal.fetch()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:DPMR:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.dpmr.symbols.hexadecimal.read()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.4 Ediagram

class EdiagramCls

Ediagram 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.vse.measurement.ediagram.clone()
```

Subgroups

6.21.1.4.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:EDIagram:CURRent
READ:VSE:MEASurement<Instance>:EDIagram:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:VSE:MEASurement<Instance>:EDIagram:CURRent
value: List[float] = driver.vse.measurement.ediagram.current.fetch()
```

Query the eye diagram current results.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
eye_diagram: Comma-separated list of 1020 frequency deviation values. The list has 60 lines with 17 values each. Each line in the diagram is interpolated from 17 values. Unit: Hz

read() → List[float]

```
# SCPI: READ:VSE:MEASurement<Instance>:EDIagram:CURRent
value: List[float] = driver.vse.measurement.ediagram.current.read()
```

Query the eye diagram current results.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
eye_diagram: Comma-separated list of 1020 frequency deviation values. The list has 60 lines with 17 values each. Each line in the diagram is interpolated from 17 values. Unit: Hz

6.21.1.5 Evm

class EvmCls

Evm 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.vse.measurement.evm.clone()
```

Subgroups

6.21.1.5.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:EVM:CURRent
READ:VSE:MEASurement<Instance>:EVM:CURRent
CALCulate:VSE:MEASurement<Instance>:EVM:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: enums.ResultStatus: Range: -999 % to 999 %, Unit: %
- Peak: enums.ResultStatus: Range: -999 deg to 999 deg, Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: float: Range: -999 % to 999 %, Unit: %
- Peak: float: Range: -999 deg to 999 deg, Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:EVM:CURRent
value: CalculateStruct = driver.vse.measurement.evm.current.calculate()
```

Query the scalar EVM results for the digital TETRA standard. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:EVM:CURRent
value: ResultData = driver.vse.measurement.evm.current.fetch()
```

Query the scalar EVM results for the digital TETRA standard. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:EVM:CURRent
value: ResultData = driver.vse.measurement.evm.current.read()
```

Query the scalar EVM results for the digital TETRA standard. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.5.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:EVM:MAXimum
READ:VSE:MEASurement<Instance>:EVM:MAXimum
CALCulate:VSE:MEASurement<Instance>:EVM:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: enums.ResultStatus: Range: -999 % to 999 %, Unit: %
- Peak: enums.ResultStatus: Range: -999 deg to 999 deg, Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: float: Range: -999 % to 999 %, Unit: %
- Peak: float: Range: -999 deg to 999 deg, Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:EVM:MAXimum
value: CalculateStruct = driver.vse.measurement.evm.maximum.calculate()
```

Query the scalar EVM results for the digital TETRA standard. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:EVM:MAXimum
value: ResultData = driver.vse.measurement.evm.maximum.fetch()
```

Query the scalar EVM results for the digital TETRA standard. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:EVM:MAXimum
value: ResultData = driver.vse.measurement.evm.maximum.read()
```

Query the scalar EVM results for the digital TETRA standard. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.6 FdError

class FdErrorCls

FdError 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.vse.measurement.fdError.clone()
```

Subgroups

6.21.1.6.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:FDERror:CURRent
READ:VSE:MEASurement<Instance>:FDERror:CURRent
CALCulate:VSE:MEASurement<Instance>:FDERror:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → ResultStatus

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:FDERror:CURRent
value: enums.ResultStatus = driver.vse.measurement.fdError.current.calculate()
```

Query the FSK deviation error results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation_error: Unit: Hz

fetch() → float

```
# SCPI: FETCh:VSE:MEASurement<Instance>:FDERror:CURRent
value: float = driver.vse.measurement.fdError.current.fetch()
```

Query the FSK deviation error results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation_error: Unit: Hz

read() → float

```
# SCPI: READ:VSE:MEASurement<Instance>:FDERror:CURRent
value: float = driver.vse.measurement.fdError.current.read()
```

Query the FSK deviation error results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation_error: Unit: Hz

6.21.1.6.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:FDERror:MAXimum
READ:VSE:MEASurement<Instance>:FDERror:MAXimum
CALCulate:VSE:MEASurement<Instance>:FDERror:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

calculate() → ResultStatus

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:FDERror:MAXimum
value: enums.ResultStatus = driver.vse.measurement.fdError.maximum.calculate()
```

Query the FSK deviation error results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation_error: Unit: Hz

fetch() → float

```
# SCPI: FETCh:VSE:MEASurement<Instance>:FDERror:MAXimum
value: float = driver.vse.measurement.fdError.maximum.fetch()
```

Query the FSK deviation error results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation_error: Unit: Hz

read() → float

```
# SCPI: READ:VSE:MEASurement<Instance>:FDERror:MAXimum
value: float = driver.vse.measurement.fdError.maximum.read()
```

Query the FSK deviation error results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation_error: Unit: Hz

6.21.1.7 Fdeviation

class FdeviationCls

Fdeviation 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.vse.measurement.fdeviation.clone()
```

Subgroups

6.21.1.7.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:FDEViation:CURRent
READ:VSE:MEASurement<Instance>:FDEViation:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:VSE:MEASurement<Instance>:FDEViation:CURRent
value: float = driver.vse.measurement.fdeviation.current.fetch()
```

Query the FSK deviation results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

read() → float

```
# SCPI: READ:VSE:MEASurement<Instance>:FDEViation:CURRent
value: float = driver.vse.measurement.fdeviation.current.read()
```

Query the FSK deviation results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

6.21.1.7.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:FDEViation:MAXimum
READ:VSE:MEASurement<Instance>:FDEViation:MAXimum
```

class MaximumCls

Maximum commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:VSE:MEASurement<Instance>:FDEViation:MAXimum
value: float = driver.vse.measurement.fdeviation.maximum.fetch()
```

Query the FSK deviation results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

read() → float

```
# SCPI: READ:VSE:MEASurement<Instance>:FDEViation:MAXimum
value: float = driver.vse.measurement.fdeviation.maximum.read()
```

Query the FSK deviation results (part of the demodulation results) .

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
fsk_deviation: Unit: Hz

6.21.1.8 FfError

class FfErrorCls

FfError 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.vse.measurement.ffError.clone()
```

Subgroups

6.21.1.8.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:FFERror:CURRent
READ:VSE:MEASurement<Instance>:FFERror:CURRent
CALCulate:VSE:MEASurement<Instance>:FFERror:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: enums.ResultStatus: Unit: %
- Peak: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: Unit: %
- Peak: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:FFERror:CURRent
value: CalculateStruct = driver.vse.measurement.ffError.current.calculate()
```

Query the FSK frequency error results (part of the demodulation results) .

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:FFERror:CURRent
value: ResultData = driver.vse.measurement.ffError.current.fetch()
```

Query the FSK frequency error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:FFERror:CURRent
value: ResultData = driver.vse.measurement.ffError.current.read()
```

Query the FSK frequency error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.8.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:FFERror:MAXimum
READ:VSE:MEASurement<Instance>:FFERror:MAXimum
CALCulate:VSE:MEASurement<Instance>:FFERror:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: enums.ResultStatus: Unit: %
- Peak: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: Unit: %
- Peak: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:FFERror:MAXimum
value: CalculateStruct = driver.vse.measurement.ffError.maximum.calculate()
```

Query the FSK frequency error results (part of the demodulation results) .

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:FFERror:MAXimum
value: ResultData = driver.vse.measurement.ffError.maximum.fetch()
```

Query the FSK frequency error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:FFERror:MAXimum
value: ResultData = driver.vse.measurement.ffError.maximum.read()
```

Query the FSK frequency error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.9 Lte

class LteCls

Lte 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.vse.measurement.lte.clone()
```

Subgroups

6.21.1.9.1 Evm

class EvmCls

Evm 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.vse.measurement.lte.evm.clone()
```

Subgroups

6.21.1.9.1.1 Current

SCPI Command:

```
READ:VSE:MEASurement<Instance>:LTE:EVM:CURRent
FETCh:VSE:MEASurement<Instance>:LTE:EVM:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Pusch_Qpsk: float: Unit: %
- Dmrs_Pusch_Qpsk: float: Unit: %
- Pucch: float: Unit: %
- Dmrs_Pucch: float: Unit: %

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:LTE:EVM:CURRent
value: ResultData = driver.vse.measurement.lte.evm.current.fetch()
```

Query LTE EVM results.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:LTE:EVM:CURRent
value: ResultData = driver.vse.measurement.lte.evm.current.read()
```

Query LTE EVM results.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.9.2 Modulation

class ModulationCls

Modulation 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.vse.measurement.lte.modulation.clone()
```

Subgroups

6.21.1.9.2.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:LTE:MODulation:CURRent
READ:VSE:MEASurement<Instance>:LTE:MODulation:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Pusch_Qpsk: float: Unit: %
- Pusch_16_Qam: float: Unit: %
- Pusch_64_Qam: float: Unit: %
- Pusch_256_Qam: float: Unit: %
- Dmrs_Pusch_Qpsk: float: Unit: %
- Dmrs_Pusch_16_Qam: float: Unit: %
- Dmrs_Pusch_64_Qam: float: Unit: %
- Dmrs_Pusch_256_Qam: float: Unit: %
- Pucch: float: Unit: %
- Dmrs_Pucch: float: Unit: %
- Power: float: Unit: dBm
- Crest_Factor: float: Unit: dB
- Evm_All: float: Unit: %
- Phys_Channel: float: Unit: %
- Phys_Signal: float: Unit: %
- Frequency_Error: float: Unit: Hz
- Sampling_Error: float: No parameter help available
- Iq_Offset: float: Unit: dB
- Iq_Gain_Imbalance: float: Unit: dB
- Iq_Quadrature_Error: float: Unit: deg

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:LTE:MODulation:CURRent
value: ResultData = driver.vse.measurement.lte.modulation.current.fetch()
```

Query LTE demodulation results.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:LTE:MODulation:CURRent
value: ResultData = driver.vse.measurement.lte.modulation.current.read()
```

Query LTE demodulation results.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.9.3 Power

class PowerCls

Power 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.vse.measurement.lte.power.clone()
```

Subgroups

6.21.1.9.3.1 Current

SCPI Command:

```
READ:VSE:MEASurement<Instance>:LTE:POWer:CURRent
FETCh:VSE:MEASurement<Instance>:LTE:POWer:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → float

```
# SCPI: FETCh:VSE:MEASurement<Instance>:LTE:POWer:CURRent
value: float = driver.vse.measurement.lte.power.current.fetch()
```

Query LTE power results.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Power value of the LTE signal Unit: dBm

read() → float

```
# SCPI: READ:VSE:MEASurement<Instance>:LTE:POWer:CURRent
value: float = driver.vse.measurement.lte.power.current.read()
```

Query LTE power results.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Power value of the LTE signal Unit: dBm

6.21.1.10 Merror

class MerrorCls

Merror 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.vse.measurement.merror.clone()
```

Subgroups

6.21.1.10.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:MERRor:CURRent
READ:VSE:MEASurement<Instance>:MERRor:CURRent
CALCulate:VSE:MEASurement<Instance>:MERRor:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: enums.ResultStatus: Unit: %
- Peak: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: Unit: %
- Peak: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:MERRor:CURRent
value: CalculateStruct = driver.vse.measurement.merror.current.calculate()
```

Query the magnitude error results (part of the demodulation results) .

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:MERRor:CURRent
value: ResultData = driver.vse.measurement.merror.current.fetch()
```

Query the magnitude error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:MERRor:CURRent
value: ResultData = driver.vse.measurement.merror.current.read()
```

Query the magnitude error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.10.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:MERRor:MAXimum
READ:VSE:MEASurement<Instance>:MERRor:MAXimum
CALCulate:VSE:MEASurement<Instance>:MERRor:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: enums.ResultStatus: Unit: %
- Peak: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Rms: float: Unit: %
- Peak: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:MERRor:MAXimum
value: CalculateStruct = driver.vse.measurement.merror.maximum.calculate()
```

Query the magnitude error results (part of the demodulation results) .

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:MERRor:MAXimum
value: ResultData = driver.vse.measurement.merror.maximum.fetch()
```

Query the magnitude error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:MERRor:MAXimum
value: ResultData = driver.vse.measurement.merror.maximum.read()
```

Query the magnitude error results (part of the demodulation results) .

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.11 Nxdn

class NxdnCls

Nxdn commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.nxdn.clone()
```

Subgroups

6.21.1.11.1 Symbols

class SymbolsCls

Symbols 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.vse.measurement.nxdn.symbols.clone()
```

Subgroups

6.21.1.11.1.1 Binary

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:NXDN:SYMBols:BINary
READ:VSE:MEASurement<Instance>:NXDN:SYMBols:BINary
```

class BinaryCls

Binary commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of binary values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:NXDN:SYMBols:BINary
value: ResultData = driver.vse.measurement.nxdn.symbols.binary.fetch()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:NXDN:SYMBols:BINary
value: ResultData = driver.vse.measurement.nxdn.symbols.binary.read()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.11.1.2 Hexadecimal

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:NXDN:SYMBols:HEXadecimal
READ:VSE:MEASurement<Instance>:NXDN:SYMBols:HEXadecimal
```

class HexadecimalCls

Hexadecimal commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of hexadecimal values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:NXDN:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.nxdn.symbols.hexadecimal.fetch()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:NXDN:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.nxdn.symbols.hexadecimal.read()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.12 Perror

class PerrorCls

Perror 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.vse.measurement.perror.clone()
```

Subgroups

6.21.1.12.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:PERRor:CURRent
READ:VSE:MEASurement<Instance>:PERRor:CURRent
CALCulate:VSE:MEASurement<Instance>:PERRor:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: enums.ResultStatus: Range: -999 deg to 999 deg, Unit: deg
- Peak: enums.ResultStatus: Range: -999 deg to 999 deg, Unit: deg

class ResultData

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: float: Range: -999 deg to 999 deg, Unit: deg
- Peak: float: Range: -999 deg to 999 deg, Unit: deg

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:PERRor:CURRent
value: CalculateStruct = driver.vse.measurement.perror.current.calculate()
```

Query the phase error results. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:PERRor:CURRent
value: ResultData = driver.vse.measurement.perror.current.fetch()
```

Query the phase error results. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:PERRor:CURRent
value: ResultData = driver.vse.measurement.perror.current.read()
```

Query the phase error results. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.12.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:PERRor:MAXimum
READ:VSE:MEASurement<Instance>:PERRor:MAXimum
CALCulate:VSE:MEASurement<Instance>:PERRor:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: enums.ResultStatus: Range: -999 deg to 999 deg, Unit: deg
- Peak: enums.ResultStatus: Range: -999 deg to 999 deg, Unit: deg

class ResultData

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: float: Range: -999 deg to 999 deg, Unit: deg
- Peak: float: Range: -999 deg to 999 deg, Unit: deg

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:PERRor:MAXimum
value: CalculateStruct = driver.vse.measurement.perror.maximum.calculate()
```

Query the phase error results. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:PERRor:MAXimum
value: ResultData = driver.vse.measurement.perror.maximum.fetch()
```

Query the phase error results. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:PERRor:MAXimum
value: ResultData = driver.vse.measurement.perror.maximum.read()
```

Query the phase error results. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.13 PowerVsTime

class PowerVsTimeCls

PowerVsTime 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.vse.measurement.powerVsTime.clone()
```

Subgroups

6.21.1.13.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:PVTime:CURRent
READ:VSE:MEASurement<Instance>:PVTime:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:VSE:MEASurement<Instance>:PVTime:CURRent
value: List[float] = driver.vse.measurement.powerVsTime.current.fetch()
```

Query the values of the power versus time diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_vs_time: Comma-separated list of power values (diagram from left to right) .
Default unit: dBm Unit: dBm

read() → List[float]

```
# SCPI: READ:VSE:MEASurement<Instance>:PVTime:CURRent
value: List[float] = driver.vse.measurement.powerVsTime.current.read()
```

Query the values of the power versus time diagram.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power_vs_time: Comma-separated list of power values (diagram from left to right) .
Default unit: dBm Unit: dBm

6.21.1.14 PtFive

class PtFiveCls

PtFive commands group definition. 6 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.ptFive.clone()
```

Subgroups

6.21.1.14.1 Mfidelity

class MfidelityCls

Mfidelity 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.vse.measurement.ptFive.mfidelity.clone()
```

Subgroups

6.21.1.14.1.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:PTFive:MFIDelity:CURRent
READ:VSE:MEASurement<Instance>:PTFive:MFIDelity:CURRent
CALCulate:VSE:MEASurement<Instance>:PTFive:MFIDelity:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: enums.ResultStatus: Unit: %
- Peak: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: float: Unit: %
- Peak: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:PTFive:MFIDelity:CURRent
value: CalculateStruct = driver.vse.measurement.ptFive.mfidelity.current.
↪calculate()
```

Query the modulation fidelity results for the digital standard 'P25'. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:PTFive:MFIDelity:CURRent
value: ResultData = driver.vse.measurement.ptFive.mfidelity.current.fetch()
```

Query the modulation fidelity results for the digital standard 'P25'. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:PTFive:MFIDelity:CURRent
value: ResultData = driver.vse.measurement.ptFive.mfidelity.current.read()
```

Query the modulation fidelity results for the digital standard 'P25'. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.14.1.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:PTFive:MFIDelity:MAXimum
READ:VSE:MEASurement<Instance>:PTFive:MFIDelity:MAXimum
CALCulate:VSE:MEASurement<Instance>:PTFive:MFIDelity:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: enums.ResultStatus: Unit: %
- Peak: enums.ResultStatus: Unit: %

class ResultData

Response structure. Fields:

- Reliability: int: No parameter help available
- Rms: float: Unit: %
- Peak: float: Unit: %

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:PTFive:MFIDelity:MAXimum
value: CalculateStruct = driver.vse.measurement.ptFive.mfidelity.maximum.
↪calculate()
```

Query the modulation fidelity results for the digital standard 'P25'. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:PTFive:MFIDelity:MAXimum
value: ResultData = driver.vse.measurement.ptFive.mfidelity.maximum.fetch()
```

Query the modulation fidelity results for the digital standard 'P25'. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:PTFive:MFIDelity:MAXimum
value: ResultData = driver.vse.measurement.ptFive.mfidelity.maximum.read()
```

Query the modulation fidelity results for the digital standard 'P25'. CALCulate commands return error indicators instead of measurement values.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.15 RfCarrier

class RfCarrierCls

RfCarrier 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.vse.measurement.rfCarrier.clone()
```

Subgroups

6.21.1.15.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:RFCarrier:CURRent
READ:VSE:MEASurement<Instance>:RFCarrier:CURRent
CALCulate:VSE:MEASurement<Instance>:RFCarrier:CURRent
```

class CurrentCls

Current commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: enums.ResultStatus: Unit: Hz
- Power_Pep: enums.ResultStatus: Unit: dBm

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: float: Unit: Hz
- Power_Pep: float: Unit: dBm
- Freq_Drift: List[float]: Unit: Hz/symbol

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:RFCarrier:CURRent
value: CalculateStruct = driver.vse.measurement.rfCarrier.current.calculate()
```

Query the RF results.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:RFCarrier:CURRent
value: ResultData = driver.vse.measurement.rfCarrier.current.fetch()
```

Query the RF results.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:RFCarrier:CURRent
value: ResultData = driver.vse.measurement.rfCarrier.current.read()
```

Query the RF results.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.15.2 Maximum

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:RFCarrier:MAXimum
READ:VSE:MEASurement<Instance>:RFCarrier:MAXimum
CALCulate:VSE:MEASurement<Instance>:RFCarrier:MAXimum
```

class MaximumCls

Maximum commands group definition. 3 total commands, 0 Subgroups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: enums.ResultStatus: Unit: Hz

- Power_Pep: enums.ResultStatus: Unit: dBm

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Freq_Error: float: Unit: Hz
- Power_Pep: float: Unit: dBm
- Freq_Drift: List[float]: Unit: Hz/symbol

calculate() → CalculateStruct

```
# SCPI: CALCulate:VSE:MEASurement<Instance>:RFCarrier:MAXimum
value: CalculateStruct = driver.vse.measurement.rfCarrier.maximum.calculate()
```

Query the RF results.

return
structure: for return value, see the help for CalculateStruct structure arguments.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:RFCarrier:MAXimum
value: ResultData = driver.vse.measurement.rfCarrier.maximum.fetch()
```

Query the RF results.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:RFCarrier:MAXimum
value: ResultData = driver.vse.measurement.rfCarrier.maximum.read()
```

Query the RF results.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.16 Sdistribute

class SdistributeCls

Sdistribute commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.sdistribute.clone()
```

Subgroups

6.21.1.16.1 Current

SCPI Command:

```
READ:VSE:MEASurement<Instance>:SDIStribute:CURRent
FETCh:VSE:MEASurement<Instance>:SDIStribute:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:VSE:MEASurement<Instance>:SDIStribute:CURRent
value: List[float] = driver.vse.measurement.sdistribute.current.fetch()
```

Query the symbol distribution that is the distribution of the measured frequency deviations.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
symbols: The list of frequency deviations.

read() → List[float]

```
# SCPI: READ:VSE:MEASurement<Instance>:SDIStribute:CURRent
value: List[float] = driver.vse.measurement.sdistribute.current.read()
```

Query the symbol distribution that is the distribution of the measured frequency deviations.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
symbols: The list of frequency deviations.

6.21.1.16.2 Xvalues

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:SDIStribute:XVALues
```

class XvaluesCls

Xvalues commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[float]

```
# SCPI: FETCh:VSE:MEASurement<Instance>:SDIStribute:XVALues
value: List[float] = driver.vse.measurement.sdistribute.xvalues.fetch()
```

Queries the symbol distribution X results.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
xvals: Unit: Hz

6.21.1.17 Spectrum

class SpectrumCls

Spectrum 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.vse.measurement.spectrum.clone()
```

Subgroups

6.21.1.17.1 Current

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:SPECtrum:CURRent
READ:VSE:MEASurement<Instance>:SPECtrum:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:VSE:MEASurement<Instance>:SPECtrum:CURRent
value: List[float] = driver.vse.measurement.spectrum.current.fetch()
```

Query the current power results of the measured spectrum.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Unit: dBm

read() → List[float]

```
# SCPI: READ:VSE:MEASurement<Instance>:SPECtrum:CURRent
value: List[float] = driver.vse.measurement.spectrum.current.read()
```

Query the current power results of the measured spectrum.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
power: Unit: dBm

6.21.1.17.2 Frequency

class FrequencyCls

Frequency 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.vse.measurement.spectrum.frequency.clone()
```

Subgroups

6.21.1.17.2.1 Start

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:SPECtrum:FREQuency:STARt
```

class StartCls

Start commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → float

```
# SCPI: FETCh:VSE:MEASurement<Instance>:SPECtrum:FREQuency:STARt
value: float = driver.vse.measurement.spectrum.frequency.start.fetch()
```

Queries the start frequency of the measured spectrum.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
start_frequency: Unit: Hz

6.21.1.17.2.2 Stop

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:SPECtrum:FREQuency:STOP
```

class StopCls

Stop commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → float

```
# SCPI: FETCh:VSE:MEASurement<Instance>:SPECtrum:FREQuency:STOP
value: float = driver.vse.measurement.spectrum.frequency.stop.fetch()
```

Queries the stop frequency of the measured spectrum.

Use RsCma.reliability.last_value to read the updated reliability indicator.

return
stop_frequency: Unit: Hz

6.21.1.18 State

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → MeasState

```
# SCPI: FETCh:VSE:MEASurement<Instance>:STATe
value: enums.MeasState = driver.vse.measurement.state.fetch()
```

Queries the main measurement state.

return
meas_state: OFF | RDY | RUN OFF Measurement is off RDY Measurement has been paused or is finished RUN Measurement is running

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.state.clone()
```

Subgroups

6.21.1.18.1 All

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[MeasState]

```
# SCPI: FETCh:VSE:MEASurement<Instance>:STATe:ALL
value: List[enums.MeasState] = driver.vse.measurement.state.all.fetch()
```

Queries the main measurement state and all substates. The substates provide additional information for the main state RUN.

return
meas_state: OFF | RUN | RDY | PENDing | ADJusted | ALIVe | FROZen | QUEued | ACTive | INValid

6.21.1.19 Tetra

class TetraCls

Tetra commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.vse.measurement.tetra.clone()
```

Subgroups

6.21.1.19.1 Symbols

class SymbolsCls

Symbols 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.vse.measurement.tetra.symbols.clone()
```

Subgroups

6.21.1.19.1.1 Binary

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:TETRa:SYMBols:BINary
READ:VSE:MEASurement<Instance>:TETRa:SYMBols:BINary
```

class BinaryCls

Binary commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of binary values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:TETRa:SYMBols:BINary
value: ResultData = driver.vse.measurement.tetra.symbols.binary.fetch()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:TETRa:SYMBols:BINary
value: ResultData = driver.vse.measurement.tetra.symbols.binary.read()
```

Query the received symbols in binary format.

return
structure: for return value, see the help for ResultData structure arguments.

6.21.1.19.1.2 Hexadecimal

SCPI Command:

```
FETCh:VSE:MEASurement<Instance>:TETRa:SYMBols:HEXadecimal
READ:VSE:MEASurement<Instance>:TETRa:SYMBols:HEXadecimal
```

class HexadecimalCls

Hexadecimal commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability indicator values'
- Symbols_Number: int: Number of values in Symbols.
- Symbols: List[str]: Comma-separated list of hexadecimal values, representing the received bit sequence. The number of values in the list equals the SymbolsNumber.

fetch() → ResultData

```
# SCPI: FETCh:VSE:MEASurement<Instance>:TETRa:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.tetra.symbols.hexadecimal.fetch()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:VSE:MEASurement<Instance>:TETRa:SYMBols:HEXadecimal
value: ResultData = driver.vse.measurement.tetra.symbols.hexadecimal.read()
```

Query the received symbols in hexadecimal format.

return
structure: for return value, see the help for ResultData structure arguments.

CHAPTER

SEVEN

RSCMA UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsCma.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsCma.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 RsCma instance with new VISA session, the session gets a new thread lock. You can assign it to other RsCma sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsCma 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

RSCMA 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

RSCMA EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

TEN

INDEX

INDEX

Symbols

A

B

C

```
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:COUPling,
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:IMPedance,
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:LEVel:DELTa:MODE,
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:LEVel:DELTa:VALue,
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:MODE,
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:SCOupling,
CONFigure:AFRF:MEASurement<Instance>:AIN<nr>:TMODe,
CONFigure:AFRF:MEASurement<Instance>:AOUT:FIRSt:LEVel,
CONFigure:AFRF:MEASurement<Instance>:AOUT:SECond:LEVel,
CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:ENABle,
CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:LEVel,
CONFigure:AFRF:MEASurement<Instance>:AOUT<nr>:SOURce,
CONFigure:AFRF:MEASurement<Instance>:CDEFinition,
CONFigure:AFRF:MEASurement<Instance>:CDEFinition:CSPace,
CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RGAMma:A,
CONFigure:AFRF:MEASurement<Instance>:CDEFinition:RGAMma:FREQuency,
CONFigure:AFRF:MEASurement<Instance>:DELTa:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DEModulation,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:MEASurement:ENAble,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:MEASurement:MODE,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:MEASurement:UPDate,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:MEASurement:USER,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:REFerence:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:REFerence:MODE,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:REFerence:UPDate,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ERRor:REFerence:USER,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:MODE,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:UPDate,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FILTer:USER,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FMSTereo:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FMSTereo:FREQuency,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:MODE,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:UPDate,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:FREQuency:USER,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:GCOupling,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:MODE,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:UPDate,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:MDEPth:USER,
CONFigure:AFRF:MEASurement<Instance>:DEModulation:TMODe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:CREPetition,
```

```
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:AFRF,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:FILTer,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:LDIRect,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:MODE,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:PATTern,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:PTYPe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:ROFactor,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:RPRBs,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:SDEViation,
CONFigure:AFRF:MEASurement<Instance>:DIGital:DCONfigure:SRATe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:LCONfigure:DMR:BERate,
CONFigure:AFRF:MEASurement<Instance>:DIGital:LCONfigure:PTFive:BERate,
CONFigure:AFRF:MEASurement<Instance>:DIGital:LCONfigure:TETRa:BERate,
CONFigure:AFRF:MEASurement<Instance>:DIGital:LCONfigure:TETRa:FERRors,
CONFigure:AFRF:MEASurement<Instance>:DIGital:LCONfigure:TTL:BERate,
CONFigure:AFRF:MEASurement<Instance>:DIGital:MCONfiguration,
CONFigure:AFRF:MEASurement<Instance>:DIGital:PCONfigure:PTYPe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:RCONfigure,
CONFigure:AFRF:MEASurement<Instance>:DIGital:RCONfigure:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DIGital:RCONfigure:BER,
CONFigure:AFRF:MEASurement<Instance>:DIGital:RCONfigure:PATTern,
CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONfigure,
CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONfigure:FILTer:DISable,
CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONfigure:FILTer:NOTCh:PATH,
CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONfigure:FREQuency:COUNter:AFR,
CONFigure:AFRF:MEASurement<Instance>:DIGital:SCONfigure:FREQuency:COUNter:AUT,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TCONfigure:BERPeriod,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TCONfigure:FREQuency:COUNter:BUR,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TCONfigure:MODE,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TCONfigure:FREQuency:COUNter:DET,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TCONfigure:TYPE,
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FEP,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:FILTer,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:LDIRect,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:MODE,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:PATTern,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:PTYPe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:ROFactor,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:RPRBs,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:SDEViation,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:SRATe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:TMODe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:
CONFigure:AFRF:MEASurement<Instance>:DIGital:TETRa:UPLink:
CONFigure:AFRF:MEASurement<Instance>:DIGital:TOUT,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:BPBits,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:BPFactor,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:CRATe,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:ENABle,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:INTerface,
CONFigure:AFRF:MEASurement<Instance>:DIGital:TTL:PATTern,
CONFigure:AFRF:MEASurement<Instance>:FILTer:DISable,
CONFigure:AFRF:MEASurement<Instance>:FILTer:NOTCh:PATH,
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AFR,
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:AUT,
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:BUR,
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:DET,
CONFigure:AFRF:MEASurement<Instance>:FREQuency:COUNter:FEP,
```

D

E

F

FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeasurement<Instance>:MEValuation:TONes:DCS:CWO
270 326
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMLeasurement<Instance>:MEValuation:TONes:DCS:DMA
271 327
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRMSurement<Instance>:MEValuation:TONes:DCS:FSK
273 328
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRMSurement<Instance>:MEValuation:TONes:DCS:FSK
275 328
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRMSurement<Instance>:MEValuation:TONes:DCS:FSK
276 329
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:DEMRMSurement<Instance>:MEValuation:TONes:DCS:FSK
278 330
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeasurement<Instance>:MEValuation:TONes:DCS:LCW
279 330
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeasurement<Instance>:MEValuation:TONes:DCS:TOC
281 331
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeasurement<Instance>:MEValuation:TONes:DEModul
282 332
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINLeasurement<Instance>:MEValuation:TONes:DEModul
284 333
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRMSurement<Instance>:MEValuation:TONes:DEModul
285 334
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRMSurement<Instance>:MEValuation:TONes:SINLeft
287 334
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRMSurement<Instance>:MEValuation:TONes:SINLeft
288 335
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:SINRMSurement<Instance>:MEValuation:TONes:SINLeft
290 336
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIPMEASurement<Instance>:MEValuation:TONes:SINRigh
291 337
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIPMEASurement<Instance>:MEValuation:TONes:SINRigh
293 338
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIPMEASurement<Instance>:MEValuation:TONes:SINRigh
294 339
FETCh:AFRF:MEASurement<Instance>:MEValuation:SQUality:VOIPMEASurement<Instance>:MEValuation:TONes:VOIP,
296 339
FETCh:AFRF:MEASurement<Instance>:MEValuation:SHATch:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:RE
318 340
FETCh:AFRF:MEASurement<Instance>:MEValuation:SHATch:AFRF:MEASurement<Instance>:MEValuation:TONes:VOIP:SE
319 341
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONEs:AFRFMEASurement<Instance>:MEValuation:VOIP:AFSignal
320 342
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONEs:AFRFMEASurement<Instance>:MEValuation:VOIP:AFSignal
321 343
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONEs:AFRFMEASurement<Instance>:MEValuation:VOIP:AFSignal
322 344
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONEs:DISTBREAsurement<Instance>:MEValuation:VOIP:AFSignal
323 345
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONEs:DISTBREAsurement<Instance>:MEValuation:VOIP:FREQuenc
324 346
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONEs:DISTBREAsurement<Instance>:MEValuation:VOIP:FREQuenc
325 347
FETCh:AFRF:MEASurement<Instance>:MEValuation:TONEs:DISTBREAsurement<Instance>:MEValuation:VOIP:FREQuenc
325 348

G

H

I

L

M

R

T

U