RsCMPX_NiotMeas

Release 5.0.70.5

Rohde & Schwarz

Apr 19, 2024

CONTENTS:


5G
CMX/P

CHAPTER

ONE

REVISION HISTORY

1.1 RsCMPX_NiotMeas

Rohde & Schwarz CMP180 Narrowband IoT Measurement RsCMPX_NiotMeas instrument driver.

Basic Hello-World code:

```
from RsCMPX_NiotMeas import *

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

Supported instruments: CMP180

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

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

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

1.1.1 Version history

Latest release notes summary: Update for FW 5.0.70

Version 5.0.70

- Update for FW 5.0.70

Version 4.0.186

- Fixed documentation

Version 4.0.185

- First released version for FW 4.0.185

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

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

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

Check out this RsCmwBase example:

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

from RsCmwBase import *  # install from pypi.org

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

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

(continues on next page)

(continued from previous page)

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

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

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

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


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


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

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

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

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

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

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

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

2.2 Installation

RsCMPX_NiotMeas is hosted on pypi.org. You can install it with pip (for example, `pip.exe` for Windows), or if you are using Pycharm (and you should be :-) direct in the Pycharm `Packet Management GUI`.

Preconditions

- Installed VISA. You can skip this if you plan to use only socket LAN connection. Download the Rohde & Schwarz VISA for Windows, Linux, Mac OS from here

Option 1 - Installing with pip.exe under Windows

- Start the command console: `WinKey + R`, type `cmd` and hit ENTER
- Change the working directory to the Python installation of your choice (adjust the user name and python version in the path):

```
cd c:\Users\John\AppData\Local\Programs\Python\Python37\Scripts
```

- Install with the command: `pip install RsCMPX_NiotMeas`

Option 2 - Installing in Pycharm

- In Pycharm Menu `File->Settings->Project->Project Interpreter` click on the '+' button on the top left (the last PyCharm version)
- Type `RsCMPX_NiotMeas` in the search box
- If you are behind a Proxy server, configure it in the Menu: `File->Settings->Appearance->System Settings->HTTP Proxy`

For more information about Rohde & Schwarz instrument remote control, check out our Instrument_Remote_Control_Web_Series .

Option 3 - Offline Installation

If you are still reading the installation chapter, it is probably because the options above did not work for you - proxy problems, your boss saw the internet bill… Here are 6 step for installing the RsCMPX_NiotMeas offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCMPX_NiotMeas needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCMPX_NiotMeas package to your computer from the pypi.org: https://pypi.org/project/RsCMPX_NiotMeas/#files to for example `c:\temp\`
- Start the command line `WinKey + R`, type `cmd` and hit ENTER
- Change the working directory to the Python installation of your choice (adjust the user name and python version in the path):

```
cd c:\Users\John\AppData\Local\Programs\Python\Python37\Scripts
```

- Install with the command: `pip install c:\temp\RsCMPX_NiotMeas-5.0.70.5.tar`

2.3 Finding Available Instruments

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

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

from RsCMPX_NiotMeas import *

# Use the instr_list string items as resource names in the RsCMPX_NiotMeas constructor
instr_list = RsCMPX_NiotMeas.list_resources("?*")
print(instr_list)
```

If you have more VISAs installed, the one actually used by default is defined by a secret widget called Visa Conflict Manager. You can force your program to use a VISA of your choice:

```
"""
Find the instruments in your environment with the defined VISA implementation
"""

from RsCMPX_NiotMeas import *

# In the optional parameter visa_select you can use for example 'rs' or 'ni'
# Rs Visa also finds any NRP-Zxx USB sensors
instr_list = RsCMPX_NiotMeas.list_resources('?*', 'rs')
print(instr_list)
```

Tip: We believe our R&S VISA is the best choice for our customers. Here are the reasons why:

- Small footprint
- Superior VXI-11 and HiSLIP performance
- Integrated legacy sensors NRP-Zxx support
- Additional VXI-11 and LXI devices search
- Availability for Windows, Linux, Mac OS

2.4 Initiating Instrument Session

RsCMPX_NiotMeas offers four different types of starting your remote-control session. We begin with the most typical case, and progress with more special ones.

Standard Session Initialization

Initiating new instrument session happens, when you instantiate the RsCMPX_NiotMeas object. Below, is a simple Hello World example. Different resource names are examples for different physical interfaces.

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

- Installed RsCMPX_NiotMeas Python module Version 5.0.70 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCMPX_NiotMeas import *

# A good practice is to assure that you have a certain minimum version installed
RsCMPX_NiotMeas.assert_minimum_version('5.0.70')
resource_string_1 = 'TCPIP::192.168.2.101::INSTR'  # Standard LAN connection (also␣
↪called VXI-11)
resource_string_2 = 'TCPIP::192.168.2.101::hislip0'  # Hi-Speed LAN connection - see␣
↪1MA208
resource_string_3 = 'GPIB::20::INSTR'  # GPIB Connection
resource_string_4 = 'USB::0x0AAD::0x0119::022019943::INSTR'  # USB-TMC (Test and␣
↪Measurement Class)

# Initializing the session
driver = RsCMPX_NiotMeas(resource_string_1)

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

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

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

Do not care about specialty of each session kind; RsCMPX_NiotMeas handles all the necessary session settings for you. You immediately have access to many identification properties in the interface `driver.utilities` . Here are same of them:

- `idn_string`

- driver_version
- visa_manufacturer
- full_instrument_model_name
- instrument_serial_number
- instrument_firmware_version
- instrument_options

The constructor also contains optional boolean arguments id_query and reset:

```
driver = RsCMPX_NiotMeas('TCPIP::192.168.56.101::hislip0', id_query=True, reset=True)
```

- Setting id_query to True (default is True) checks, whether your instrument can be used with the RsCMPX_NiotMeas module.
- Setting reset to True (default is False) resets your instrument. It is equivalent to calling the reset() method.

Selecting a Specific VISA

Just like in the function list_resources(), the RsCMPX_NiotMeas allows you to choose which VISA to use:

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

from RsCMPX_NiotMeas import *

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

idn = driver.utilities.query_str('*IDN?')
print(f"\nHello, I am: '{idn}'")
print(f"\nI am using the VISA from: {driver.utilities.visa_manufacturer}")

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

No VISA Session

We recommend using VISA when possible preferrably with HiSlip session because of its low latency. However, if you are a strict VISA denier, RsCMPX_NiotMeas has something for you too - **no Visa installation raw LAN socket**:

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

from RsCMPX_NiotMeas import *

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

(continues on next page)

(continued from previous page)

```
print(f"\nHello, I am: '{driver.utilities.idn_string}'")

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

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

Simulating Session

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

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

More `option_string` tokens are separated by comma:

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

Shared Session

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

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

from RsCMPX_NiotMeas import *

driver1 = RsCMPX_NiotMeas('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsCMPX_NiotMeas.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 RsCMPX_NiotMeas API Structure. If for any reason you want to use the plain SCPI, use the `utilities` interface's two basic methods:

- `write_str()` - writing a command without an answer, for example ***RST**
- `query_str()` - querying your instrument, for example the ***IDN?** query

You may ask a question. Actually, two questions:

- **Q1**: Why there are not called `write()` and `query()` ?
- **Q2**: Where is the `read()` ?

Answer 1: Actually, there are - the `write_str()` / `write()` and `query_str()` / `query()` are aliases, and you can use any of them. We promote the `_str` names, to clearly show you want to work with strings. Strings in Python3 are Unicode, the *bytes* and *string* objects are not interchangeable, since one character might be represented by more than 1 byte. To avoid mixing string and binary communication, all the method names for binary transfer contain `_bin` in the name.

Answer 2: Short answer - you do not need it. Long answer - your instrument never sends unsolicited responses. If you send a set command, you use `write_str()`. For a query command, you use `query_str()`. So, you really do not need it...

Bottom line - if you are used to `write()` and `query()` methods, from pyvisa, the `write_str()` and `query_str()` are their equivalents.

Enough with the theory, let us look at an example. Simple write, and query:

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

from RsCMPX_NiotMeas import *

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

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

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

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

from RsCMPX_NiotMeas import *

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

(continues on next page)

(continued from previous page)

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

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

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

After this time, the RsCMPX_NiotMeas raises an exception. Speaking of exceptions, an important feature of the RsCMPX_NiotMeas is **Instrument Status Checking**. Check out the next chapter that describes the error checking in details.

For completion, we mention other string-based `write_xxx()` and `query_xxx()` methods - all in one example. They are convenient extensions providing type-safe float/boolean/integer setting/querying features:

```
"""
Basic string write_xxx / query_xxx
"""

from RsCMPX_NiotMeas import *

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

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

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

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

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

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

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

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

2.6 Error Checking

RsCMPX_NiotMeas pushes limits even further (internal R&S joke): It has a built-in mechanism that after each command/query checks the instrument's status subsystem, and raises an exception if it detects an error. For those who are already screaming: **Speed Performance Penalty!!!**, don't worry, you can disable it.

Instrument status checking is very useful since in case your command/query caused an error, you are immediately informed about it. Status checking has in most cases no practical effect on the speed performance of your program. However, if for example, you do many repetitions of short write/query sequences, it might make a difference to switch it off:

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

To clear the instrument status subsystem of all errors, call this method:

```
driver.utilities.clear_status()
```

Instrument's status system error queue is clear-on-read. It means, if you query its content, you clear it at the same time. To query and clear list of all the current errors, use this snippet:

```
errors_list = driver.utilities.query_all_errors()
```

See the next chapter on how to react on errors.

2.7 Exception Handling

The base class for all the exceptions raised by the RsCMPX_NiotMeas is `RsInstrException`. Inherited exception classes:

- `ResourceError` raised in the constructor by problems with initiating the instrument, for example wrong or non-existing resource name
- `StatusException` raised if a command or a query generated error in the instrument's error queue
- `TimeoutException` raised if a visa timeout or an opc timeout is reached

In this example we show usage of all of them. Because it is difficult to generate an error using the instrument-specific SCPI API, we use plain SCPI commands:

```
"""
Showing how to deal with exceptions
"""

from RsCMPX_NiotMeas import *

```

(continues on next page)

(continued from previous page)

```
driver = None
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsCMPX_NiotMeas('TCPIP::10.112.1.179::hislip0')
except ResourceError as e:
    print(e.args[0])
    print('Your instrument is probably OFF...')
    # Exit now, no point of continuing
    exit(1)

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

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

except StatusException as e:
    # Instrument status error
    print(e.args[0])
    print('Nothing to see here, moving on...')

except TimeoutException as e:
    # Timeout error
    print(e.args[0])
    print('That took a long time...')

except RsInstrException as e:
    # RsInstrException is a base class for all the RsCMPX_NiotMeas exceptions
    print(e.args[0])
    print('Some other RsCMPX_NiotMeas error...')

finally:
    driver.utilities.visa_timeout = 5000
    # Close the session in any case
    driver.close()
```

Tip: General rules for exception handling:

- If you are sending commands that might generate errors in the instrument, for example deleting a file which does not exist, use the **OPTION 1** - temporarily disable status checking, send the command, clear the error queue and enable the status checking again.
- If you are sending queries that might generate errors or timeouts, for example querying measurement that can not be performed at the moment, use the **OPTION 2** - try/except with optionally adjusting the timeouts.

2.8 Transferring Files

Instrument -> PC

You definitely experienced it: you just did a perfect measurement, saved the results as a screenshot to an instrument's storage drive. Now you want to transfer it to your PC. With RsCMPX_NiotMeas, no problem, just figure out where the screenshot was stored on the instrument. In our case, it is */var/user/instr_screenshot.png*:

```
driver.utilities.read_file_from_instrument_to_pc(
    r'/var/user/instr_screenshot.png',
    r'c:\temp\pc_screenshot.png')
```

PC -> Instrument

Another common scenario: Your cool test program contains a setup file you want to transfer to your instrument: Here is the RsCMPX_NiotMeas one-liner split into 3 lines:

```
driver.utilities.send_file_from_pc_to_instrument(
    r'c:\MyCoolTestProgram\instr_setup.sav',
    r'/var/appdata/instr_setup.sav')
```

2.9 Writing Binary Data

Writing from bytes

An example where you need to send binary data is a waveform file of a vector signal generator. First, you compose your wform_data as `bytes`, and then you send it with `write_bin_block()`:

```
# MyWaveform.wv is an instrument file name under which this data is stored
driver.utilities.write_bin_block(
    "SOUR:BB:ARB:WAV:DATA 'MyWaveform.wv',",
    wform_data)
```

Note: Notice the `write_bin_block()` has two parameters:

- `string` parameter `cmd` for the SCPI command
- `bytes` parameter `payload` for the actual binary data to send

Writing from PC files

Similar to querying binary data to a file, you can write binary data from a file. The second parameter is then the PC file path the content of which you want to send:

```
driver.utilities.write_bin_block_from_file(
    "SOUR:BB:ARB:WAV:DATA 'MyWaveform.wv',",
    r"c:\temp\wform_data.wv")
```

2.10 Transferring Big Data with Progress

We can agree that it can be annoying using an application that shows no progress for long-lasting operations. The same is true for remote-control programs. Luckily, the RsCMPX_NiotMeas has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

RsCMPX_NiotMeas allows you to register a function (programmers fancy name is `callback`), which is then periodicaly invoked after transfer of one data chunk. You can define that chunk size, which gives you control over the callback invoke frequency. You can even slow down the transfer speed, if you want to process the data as they arrive (direction instrument -> PC).

To show this in praxis, we are going to use another *University-Professor-Example*: querying the ***IDN?** with chunk size of 2 bytes and delay of 200ms between each chunk read:

```
"""
Event handlers by reading
"""

from RsCMPX_NiotMeas import *
import time


def my_transfer_handler(args):
    """Function called each time a chunk of data is transferred"""
    # Total size is not always known at the beginning of the transfer
    total_size = args.total_size if args.total_size is not None else "unknown"

    print(f"Context: '{args.context}{'with opc' if args.opc_sync else ''}', "
        f"chunk {args.chunk_ix}, "
        f"transferred {args.transferred_size} bytes, "
        f"total size {total_size}, "
        f"direction {'reading' if args.reading else 'writing'}, "
        f"data '{args.data}'")

    if args.end_of_transfer:
        print('End of Transfer')
    time.sleep(0.2)


driver = RsCMPX_NiotMeas('TCPIP::192.168.56.101::INSTR')

driver.events.on_read_handler = my_transfer_handler
# Switch on the data to be included in the event arguments
```

(continues on next page)

(continued from previous page)

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

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

If you start it, you might wonder (or maybe not): why is the `args.total_size = None`? The reason is, in this particular case the RsCMPX_NiotMeas does not know the size of the complete response up-front. However, if you use the same mechanism for transfer of a known data size (for example, file transfer), you get the information about the total size too, and hence you can calculate the progress as:

*progress [pct] = 100 * args.transferred_size / args.total_size*

Snippet of transferring file from PC to instrument, the rest of the code is the same as in the previous example:

```
driver.events.on_write_handler = my_transfer_handler
driver.events.io_events_include_data = True
driver.data_chunk_size = 1000
driver.utilities.send_file_from_pc_to_instrument(
    r'c:\MyCoolTestProgram\my_big_file.bin',
    r'/var/user/my_big_file.bin')
# Unregister the event handler
driver.events.on_write_handler = None
```

2.11 Multithreading

You are at the party, many people talking over each other. Not every person can deal with such crosstalk, neither can measurement instruments. For this reason, RsCMPX_NiotMeas has a feature of scheduling the access to your instrument by using so-called **Locks**. Locks make sure that there can be just one client at a time *talking* to your instrument. Talking in this context means completing one communication step - one command write or write/read or write/read/error check.

To describe how it works, and where it matters, we take three typical mulithread scenarios:

One instrument session, accessed from multiple threads

You are all set - the lock is a part of your instrument session. Check out the following example - it will execute properly, although the instrument gets 10 queries at the same time:

```
"""
Multiple threads are accessing one RsCMPX_NiotMeas object
"""

import threading
from RsCMPX_NiotMeas import *
```

(continues on next page)

(continued from previous page)

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


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

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsCMPX_NiotMeas import *


def execute(session: RsCMPX_NiotMeas, session_ix, index) -> None:
    """Executed in a separate thread."""
    print(f'{index} session {session_ix} query start...')
    session.utilities.query_str('*IDN?')
    print(f'{index} session {session_ix} query end')


driver1 = RsCMPX_NiotMeas('TCPIP::192.168.56.101::INSTR')
driver2 = RsCMPX_NiotMeas.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()
```

(continues on next page)

(continued from previous page)

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

Run the following example:

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

import threading
from RsCMPX_NiotMeas import *


def execute(session: RsCMPX_NiotMeas, session_ix, index) -> None:
    """Executed in a separate thread."""
    print(f'{index} session {session_ix} query start...')
    session.utilities.query_str('*IDN?')
    print(f'{index} session {session_ix} query end')


driver1 = RsCMPX_NiotMeas('TCPIP::192.168.56.101::INSTR')
driver2 = RsCMPX_NiotMeas('TCPIP::192.168.56.101::INSTR')
```

(continues on next page)

(continued from previous page)

```
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 RsCMPX_NiotMeas import *
```

(continues on next page)

(continued from previous page)

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

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

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

from RsCMPX_NiotMeas import *

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

(continues on next page)

(continued from previous page)

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

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

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

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

driver.utilities.reset()

# Close the session
driver.close()

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

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
from RsCMPX_NiotMeas import *

driver = RsCMPX_NiotMeas('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 Band

```
# First value:
value = enums.Band.OB1
# Last value:
value = enums.Band.UDEF
# All values (39x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB17 | OB18
OB19 | OB2 | OB20 | OB21 | OB22 | OB23 | OB24 | OB25
OB255 | OB256 | OB26 | OB27 | OB28 | OB3 | OB30 | OB31
OB4 | OB5 | OB65 | OB66 | OB68 | OB7 | OB70 | OB71
OB72 | OB73 | OB74 | OB8 | OB85 | OB9 | UDEF
```

3.2 ChannelBw

```
# Example value:
value = enums.ChannelBw.B200
# All values (1x):
B200
```

3.3 CmwsConnector

```
# First value:
value = enums.CmwsConnector.R11
# Last value:
value = enums.CmwsConnector.RB8
# All values (48x):
R11 | R12 | R13 | R14 | R15 | R16 | R17 | R18
R21 | R22 | R23 | R24 | R25 | R26 | R27 | R28
R31 | R32 | R33 | R34 | R35 | R36 | R37 | R38
R41 | R42 | R43 | R44 | R45 | R46 | R47 | R48
RA1 | RA2 | RA3 | RA4 | RA5 | RA6 | RA7 | RA8
RB1 | RB2 | RB3 | RB4 | RB5 | RB6 | RB7 | RB8
```

3.4 CyclicPrefix

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

3.5 LeadLag

```
# Example value:
value = enums.LeadLag.OFF
# All values (2x):
OFF | S1
```

3.6 ListMode

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

3.7 LowHigh

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

3.8 MeasurementMode

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

3.9 Mode

```
# Example value:
value = enums.Mode.FDD
# All values (2x):
FDD | TDD
```

3.10 ModScheme

```
# Example value:
value = enums.ModScheme.BPSK
# All values (3x):
BPSK | Q16 | QPSK
```

3.11 NofRepetitions

```
# First value:
value = enums.NofRepetitions.NR1
# Last value:
value = enums.NofRepetitions.NR8
# All values (12x):
NR1 | NR128 | NR16 | NR1K | NR2 | NR256 | NR2K | NR32
NR4 | NR512 | NR64 | NR8
```

3.12 NofRepetitionsList

```
# First value:
value = enums.NofRepetitionsList.NR1
# Last value:
value = enums.NofRepetitionsList.NR8
# All values (16x):
NR1 | NR128 | NR1536 | NR16 | NR192 | NR1K | NR2 | NR256
NR2K | NR32 | NR384 | NR4 | NR512 | NR64 | NR768 | NR8
```

3.13 NofRsrcUnits

```
# Example value:
value = enums.NofRsrcUnits.NRU01
# All values (8x):
NRU01 | NRU02 | NRU03 | NRU04 | NRU05 | NRU06 | NRU08 | NRU10
```

3.14 NpuschFormat

```
# Example value:
value = enums.NpuschFormat.F1
# All values (2x):
F1 | F2
```

3.15 ObwMode

```
# Example value:
value = enums.ObwMode.BW99
# All values (2x):
BW99 | M26
```

3.16 ParameterSetMode

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

3.17 PeriodPreamble

```
# Example value:
value = enums.PeriodPreamble.MS160
# All values (5x):
MS160 | MS240 | MS320 | MS40 | MS80
```

3.18 Repeat

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

3.19 ResourceState

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

3.20 ResultStatus2

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

3.21 RetriggerFlag

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

3.22 SignalSlope

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

3.23 StopCondition

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

3.24 SubCarrSpacing

```
# Example value:
value = enums.SubCarrSpacing.S15K
# All values (2x):
S15K | S3K75
```

3.25 TargetStateA

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

3.26 TargetSyncState

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

3.27 TimeMask

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

CHAPTER

FOUR

REPCAPS

4.1 Instance (Global)

```
# Setting:
driver.repcap_instance_set(repcap.Instance.Inst1)
# Range:
Inst1 .. Inst16
# All values (16x):
Inst1 | Inst2 | Inst3 | Inst4 | Inst5 | Inst6 | Inst7 | Inst8
Inst9 | Inst10 | Inst11 | Inst12 | Inst13 | Inst14 | Inst15 | Inst16
```

4.2 Limit

```
# First value:
value = repcap.Limit.Nr1
# Range:
Nr1 .. Nr12
# All values (12x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12
```

4.3 Preamble

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

4.4 PreambleFormat

```
# First value:
value = repcap.PreambleFormat.Fmt1
# Values (2x):
Fmt1 | Fmt2
```

4.5 Segment

```
# First value:
value = repcap.Segment.Nr1
# Range:
Nr1 .. Nr2000
# All values (2000x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23 | Nr24
Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31 | Nr32
Nr33 | Nr34 | Nr35 | Nr36 | Nr37 | Nr38 | Nr39 | Nr40
Nr41 | Nr42 | Nr43 | Nr44 | Nr45 | Nr46 | Nr47 | Nr48
Nr49 | Nr50 | Nr51 | Nr52 | Nr53 | Nr54 | Nr55 | Nr56
Nr57 | Nr58 | Nr59 | Nr60 | Nr61 | Nr62 | Nr63 | Nr64
Nr65 | Nr66 | Nr67 | Nr68 | Nr69 | Nr70 | Nr71 | Nr72
Nr73 | Nr74 | Nr75 | Nr76 | Nr77 | Nr78 | Nr79 | Nr80
Nr81 | Nr82 | Nr83 | Nr84 | Nr85 | Nr86 | Nr87 | Nr88
Nr89 | Nr90 | Nr91 | Nr92 | Nr93 | Nr94 | Nr95 | Nr96
Nr97 | Nr98 | Nr99 | Nr100 | Nr101 | Nr102 | Nr103 | Nr104
Nr105 | Nr106 | Nr107 | Nr108 | Nr109 | Nr110 | Nr111 | Nr112
Nr113 | Nr114 | Nr115 | Nr116 | Nr117 | Nr118 | Nr119 | Nr120
Nr121 | Nr122 | Nr123 | Nr124 | Nr125 | Nr126 | Nr127 | Nr128
Nr129 | Nr130 | Nr131 | Nr132 | Nr133 | Nr134 | Nr135 | Nr136
Nr137 | Nr138 | Nr139 | Nr140 | Nr141 | Nr142 | Nr143 | Nr144
Nr145 | Nr146 | Nr147 | Nr148 | Nr149 | Nr150 | Nr151 | Nr152
Nr153 | Nr154 | Nr155 | Nr156 | Nr157 | Nr158 | Nr159 | Nr160
Nr161 | Nr162 | Nr163 | Nr164 | Nr165 | Nr166 | Nr167 | Nr168
Nr169 | Nr170 | Nr171 | Nr172 | Nr173 | Nr174 | Nr175 | Nr176
Nr177 | Nr178 | Nr179 | Nr180 | Nr181 | Nr182 | Nr183 | Nr184
Nr185 | Nr186 | Nr187 | Nr188 | Nr189 | Nr190 | Nr191 | Nr192
Nr193 | Nr194 | Nr195 | Nr196 | Nr197 | Nr198 | Nr199 | Nr200
Nr201 | Nr202 | Nr203 | Nr204 | Nr205 | Nr206 | Nr207 | Nr208
Nr209 | Nr210 | Nr211 | Nr212 | Nr213 | Nr214 | Nr215 | Nr216
Nr217 | Nr218 | Nr219 | Nr220 | Nr221 | Nr222 | Nr223 | Nr224
Nr225 | Nr226 | Nr227 | Nr228 | Nr229 | Nr230 | Nr231 | Nr232
Nr233 | Nr234 | Nr235 | Nr236 | Nr237 | Nr238 | Nr239 | Nr240
Nr241 | Nr242 | Nr243 | Nr244 | Nr245 | Nr246 | Nr247 | Nr248
Nr249 | Nr250 | Nr251 | Nr252 | Nr253 | Nr254 | Nr255 | Nr256
Nr257 | Nr258 | Nr259 | Nr260 | Nr261 | Nr262 | Nr263 | Nr264
Nr265 | Nr266 | Nr267 | Nr268 | Nr269 | Nr270 | Nr271 | Nr272
Nr273 | Nr274 | Nr275 | Nr276 | Nr277 | Nr278 | Nr279 | Nr280
Nr281 | Nr282 | Nr283 | Nr284 | Nr285 | Nr286 | Nr287 | Nr288
```

(continues on next page)

(continued from previous page)

```
Nr289 | Nr290 | Nr291 | Nr292 | Nr293 | Nr294 | Nr295 | Nr296
Nr297 | Nr298 | Nr299 | Nr300 | Nr301 | Nr302 | Nr303 | Nr304
Nr305 | Nr306 | Nr307 | Nr308 | Nr309 | Nr310 | Nr311 | Nr312
Nr313 | Nr314 | Nr315 | Nr316 | Nr317 | Nr318 | Nr319 | Nr320
Nr321 | Nr322 | Nr323 | Nr324 | Nr325 | Nr326 | Nr327 | Nr328
Nr329 | Nr330 | Nr331 | Nr332 | Nr333 | Nr334 | Nr335 | Nr336
Nr337 | Nr338 | Nr339 | Nr340 | Nr341 | Nr342 | Nr343 | Nr344
Nr345 | Nr346 | Nr347 | Nr348 | Nr349 | Nr350 | Nr351 | Nr352
Nr353 | Nr354 | Nr355 | Nr356 | Nr357 | Nr358 | Nr359 | Nr360
Nr361 | Nr362 | Nr363 | Nr364 | Nr365 | Nr366 | Nr367 | Nr368
Nr369 | Nr370 | Nr371 | Nr372 | Nr373 | Nr374 | Nr375 | Nr376
Nr377 | Nr378 | Nr379 | Nr380 | Nr381 | Nr382 | Nr383 | Nr384
Nr385 | Nr386 | Nr387 | Nr388 | Nr389 | Nr390 | Nr391 | Nr392
Nr393 | Nr394 | Nr395 | Nr396 | Nr397 | Nr398 | Nr399 | Nr400
Nr401 | Nr402 | Nr403 | Nr404 | Nr405 | Nr406 | Nr407 | Nr408
Nr409 | Nr410 | Nr411 | Nr412 | Nr413 | Nr414 | Nr415 | Nr416
Nr417 | Nr418 | Nr419 | Nr420 | Nr421 | Nr422 | Nr423 | Nr424
Nr425 | Nr426 | Nr427 | Nr428 | Nr429 | Nr430 | Nr431 | Nr432
Nr433 | Nr434 | Nr435 | Nr436 | Nr437 | Nr438 | Nr439 | Nr440
Nr441 | Nr442 | Nr443 | Nr444 | Nr445 | Nr446 | Nr447 | Nr448
Nr449 | Nr450 | Nr451 | Nr452 | Nr453 | Nr454 | Nr455 | Nr456
Nr457 | Nr458 | Nr459 | Nr460 | Nr461 | Nr462 | Nr463 | Nr464
Nr465 | Nr466 | Nr467 | Nr468 | Nr469 | Nr470 | Nr471 | Nr472
Nr473 | Nr474 | Nr475 | Nr476 | Nr477 | Nr478 | Nr479 | Nr480
Nr481 | Nr482 | Nr483 | Nr484 | Nr485 | Nr486 | Nr487 | Nr488
Nr489 | Nr490 | Nr491 | Nr492 | Nr493 | Nr494 | Nr495 | Nr496
Nr497 | Nr498 | Nr499 | Nr500 | Nr501 | Nr502 | Nr503 | Nr504
Nr505 | Nr506 | Nr507 | Nr508 | Nr509 | Nr510 | Nr511 | Nr512
Nr513 | Nr514 | Nr515 | Nr516 | Nr517 | Nr518 | Nr519 | Nr520
Nr521 | Nr522 | Nr523 | Nr524 | Nr525 | Nr526 | Nr527 | Nr528
Nr529 | Nr530 | Nr531 | Nr532 | Nr533 | Nr534 | Nr535 | Nr536
Nr537 | Nr538 | Nr539 | Nr540 | Nr541 | Nr542 | Nr543 | Nr544
Nr545 | Nr546 | Nr547 | Nr548 | Nr549 | Nr550 | Nr551 | Nr552
Nr553 | Nr554 | Nr555 | Nr556 | Nr557 | Nr558 | Nr559 | Nr560
Nr561 | Nr562 | Nr563 | Nr564 | Nr565 | Nr566 | Nr567 | Nr568
Nr569 | Nr570 | Nr571 | Nr572 | Nr573 | Nr574 | Nr575 | Nr576
Nr577 | Nr578 | Nr579 | Nr580 | Nr581 | Nr582 | Nr583 | Nr584
Nr585 | Nr586 | Nr587 | Nr588 | Nr589 | Nr590 | Nr591 | Nr592
Nr593 | Nr594 | Nr595 | Nr596 | Nr597 | Nr598 | Nr599 | Nr600
Nr601 | Nr602 | Nr603 | Nr604 | Nr605 | Nr606 | Nr607 | Nr608
Nr609 | Nr610 | Nr611 | Nr612 | Nr613 | Nr614 | Nr615 | Nr616
Nr617 | Nr618 | Nr619 | Nr620 | Nr621 | Nr622 | Nr623 | Nr624
Nr625 | Nr626 | Nr627 | Nr628 | Nr629 | Nr630 | Nr631 | Nr632
Nr633 | Nr634 | Nr635 | Nr636 | Nr637 | Nr638 | Nr639 | Nr640
Nr641 | Nr642 | Nr643 | Nr644 | Nr645 | Nr646 | Nr647 | Nr648
Nr649 | Nr650 | Nr651 | Nr652 | Nr653 | Nr654 | Nr655 | Nr656
Nr657 | Nr658 | Nr659 | Nr660 | Nr661 | Nr662 | Nr663 | Nr664
Nr665 | Nr666 | Nr667 | Nr668 | Nr669 | Nr670 | Nr671 | Nr672
Nr673 | Nr674 | Nr675 | Nr676 | Nr677 | Nr678 | Nr679 | Nr680
Nr681 | Nr682 | Nr683 | Nr684 | Nr685 | Nr686 | Nr687 | Nr688
Nr689 | Nr690 | Nr691 | Nr692 | Nr693 | Nr694 | Nr695 | Nr696
Nr697 | Nr698 | Nr699 | Nr700 | Nr701 | Nr702 | Nr703 | Nr704
```

(continues on next page)

(continued from previous page)

```
Nr705 | Nr706 | Nr707 | Nr708 | Nr709 | Nr710 | Nr711 | Nr712
Nr713 | Nr714 | Nr715 | Nr716 | Nr717 | Nr718 | Nr719 | Nr720
Nr721 | Nr722 | Nr723 | Nr724 | Nr725 | Nr726 | Nr727 | Nr728
Nr729 | Nr730 | Nr731 | Nr732 | Nr733 | Nr734 | Nr735 | Nr736
Nr737 | Nr738 | Nr739 | Nr740 | Nr741 | Nr742 | Nr743 | Nr744
Nr745 | Nr746 | Nr747 | Nr748 | Nr749 | Nr750 | Nr751 | Nr752
Nr753 | Nr754 | Nr755 | Nr756 | Nr757 | Nr758 | Nr759 | Nr760
Nr761 | Nr762 | Nr763 | Nr764 | Nr765 | Nr766 | Nr767 | Nr768
Nr769 | Nr770 | Nr771 | Nr772 | Nr773 | Nr774 | Nr775 | Nr776
Nr777 | Nr778 | Nr779 | Nr780 | Nr781 | Nr782 | Nr783 | Nr784
Nr785 | Nr786 | Nr787 | Nr788 | Nr789 | Nr790 | Nr791 | Nr792
Nr793 | Nr794 | Nr795 | Nr796 | Nr797 | Nr798 | Nr799 | Nr800
Nr801 | Nr802 | Nr803 | Nr804 | Nr805 | Nr806 | Nr807 | Nr808
Nr809 | Nr810 | Nr811 | Nr812 | Nr813 | Nr814 | Nr815 | Nr816
Nr817 | Nr818 | Nr819 | Nr820 | Nr821 | Nr822 | Nr823 | Nr824
Nr825 | Nr826 | Nr827 | Nr828 | Nr829 | Nr830 | Nr831 | Nr832
Nr833 | Nr834 | Nr835 | Nr836 | Nr837 | Nr838 | Nr839 | Nr840
Nr841 | Nr842 | Nr843 | Nr844 | Nr845 | Nr846 | Nr847 | Nr848
Nr849 | Nr850 | Nr851 | Nr852 | Nr853 | Nr854 | Nr855 | Nr856
Nr857 | Nr858 | Nr859 | Nr860 | Nr861 | Nr862 | Nr863 | Nr864
Nr865 | Nr866 | Nr867 | Nr868 | Nr869 | Nr870 | Nr871 | Nr872
Nr873 | Nr874 | Nr875 | Nr876 | Nr877 | Nr878 | Nr879 | Nr880
Nr881 | Nr882 | Nr883 | Nr884 | Nr885 | Nr886 | Nr887 | Nr888
Nr889 | Nr890 | Nr891 | Nr892 | Nr893 | Nr894 | Nr895 | Nr896
Nr897 | Nr898 | Nr899 | Nr900 | Nr901 | Nr902 | Nr903 | Nr904
Nr905 | Nr906 | Nr907 | Nr908 | Nr909 | Nr910 | Nr911 | Nr912
Nr913 | Nr914 | Nr915 | Nr916 | Nr917 | Nr918 | Nr919 | Nr920
Nr921 | Nr922 | Nr923 | Nr924 | Nr925 | Nr926 | Nr927 | Nr928
Nr929 | Nr930 | Nr931 | Nr932 | Nr933 | Nr934 | Nr935 | Nr936
Nr937 | Nr938 | Nr939 | Nr940 | Nr941 | Nr942 | Nr943 | Nr944
Nr945 | Nr946 | Nr947 | Nr948 | Nr949 | Nr950 | Nr951 | Nr952
Nr953 | Nr954 | Nr955 | Nr956 | Nr957 | Nr958 | Nr959 | Nr960
Nr961 | Nr962 | Nr963 | Nr964 | Nr965 | Nr966 | Nr967 | Nr968
Nr969 | Nr970 | Nr971 | Nr972 | Nr973 | Nr974 | Nr975 | Nr976
Nr977 | Nr978 | Nr979 | Nr980 | Nr981 | Nr982 | Nr983 | Nr984
Nr985 | Nr986 | Nr987 | Nr988 | Nr989 | Nr990 | Nr991 | Nr992
Nr993 | Nr994 | Nr995 | Nr996 | Nr997 | Nr998 | Nr999 | Nr1000
Nr1001 | Nr1002 | Nr1003 | Nr1004 | Nr1005 | Nr1006 | Nr1007 | Nr1008
Nr1009 | Nr1010 | Nr1011 | Nr1012 | Nr1013 | Nr1014 | Nr1015 | Nr1016
Nr1017 | Nr1018 | Nr1019 | Nr1020 | Nr1021 | Nr1022 | Nr1023 | Nr1024
Nr1025 | Nr1026 | Nr1027 | Nr1028 | Nr1029 | Nr1030 | Nr1031 | Nr1032
Nr1033 | Nr1034 | Nr1035 | Nr1036 | Nr1037 | Nr1038 | Nr1039 | Nr1040
Nr1041 | Nr1042 | Nr1043 | Nr1044 | Nr1045 | Nr1046 | Nr1047 | Nr1048
Nr1049 | Nr1050 | Nr1051 | Nr1052 | Nr1053 | Nr1054 | Nr1055 | Nr1056
Nr1057 | Nr1058 | Nr1059 | Nr1060 | Nr1061 | Nr1062 | Nr1063 | Nr1064
Nr1065 | Nr1066 | Nr1067 | Nr1068 | Nr1069 | Nr1070 | Nr1071 | Nr1072
Nr1073 | Nr1074 | Nr1075 | Nr1076 | Nr1077 | Nr1078 | Nr1079 | Nr1080
Nr1081 | Nr1082 | Nr1083 | Nr1084 | Nr1085 | Nr1086 | Nr1087 | Nr1088
Nr1089 | Nr1090 | Nr1091 | Nr1092 | Nr1093 | Nr1094 | Nr1095 | Nr1096
Nr1097 | Nr1098 | Nr1099 | Nr1100 | Nr1101 | Nr1102 | Nr1103 | Nr1104
Nr1105 | Nr1106 | Nr1107 | Nr1108 | Nr1109 | Nr1110 | Nr1111 | Nr1112
Nr1113 | Nr1114 | Nr1115 | Nr1116 | Nr1117 | Nr1118 | Nr1119 | Nr1120
```

(continues on next page)

(continued from previous page)

```
Nr1121 | Nr1122 | Nr1123 | Nr1124 | Nr1125 | Nr1126 | Nr1127 | Nr1128
Nr1129 | Nr1130 | Nr1131 | Nr1132 | Nr1133 | Nr1134 | Nr1135 | Nr1136
Nr1137 | Nr1138 | Nr1139 | Nr1140 | Nr1141 | Nr1142 | Nr1143 | Nr1144
Nr1145 | Nr1146 | Nr1147 | Nr1148 | Nr1149 | Nr1150 | Nr1151 | Nr1152
Nr1153 | Nr1154 | Nr1155 | Nr1156 | Nr1157 | Nr1158 | Nr1159 | Nr1160
Nr1161 | Nr1162 | Nr1163 | Nr1164 | Nr1165 | Nr1166 | Nr1167 | Nr1168
Nr1169 | Nr1170 | Nr1171 | Nr1172 | Nr1173 | Nr1174 | Nr1175 | Nr1176
Nr1177 | Nr1178 | Nr1179 | Nr1180 | Nr1181 | Nr1182 | Nr1183 | Nr1184
Nr1185 | Nr1186 | Nr1187 | Nr1188 | Nr1189 | Nr1190 | Nr1191 | Nr1192
Nr1193 | Nr1194 | Nr1195 | Nr1196 | Nr1197 | Nr1198 | Nr1199 | Nr1200
Nr1201 | Nr1202 | Nr1203 | Nr1204 | Nr1205 | Nr1206 | Nr1207 | Nr1208
Nr1209 | Nr1210 | Nr1211 | Nr1212 | Nr1213 | Nr1214 | Nr1215 | Nr1216
Nr1217 | Nr1218 | Nr1219 | Nr1220 | Nr1221 | Nr1222 | Nr1223 | Nr1224
Nr1225 | Nr1226 | Nr1227 | Nr1228 | Nr1229 | Nr1230 | Nr1231 | Nr1232
Nr1233 | Nr1234 | Nr1235 | Nr1236 | Nr1237 | Nr1238 | Nr1239 | Nr1240
Nr1241 | Nr1242 | Nr1243 | Nr1244 | Nr1245 | Nr1246 | Nr1247 | Nr1248
Nr1249 | Nr1250 | Nr1251 | Nr1252 | Nr1253 | Nr1254 | Nr1255 | Nr1256
Nr1257 | Nr1258 | Nr1259 | Nr1260 | Nr1261 | Nr1262 | Nr1263 | Nr1264
Nr1265 | Nr1266 | Nr1267 | Nr1268 | Nr1269 | Nr1270 | Nr1271 | Nr1272
Nr1273 | Nr1274 | Nr1275 | Nr1276 | Nr1277 | Nr1278 | Nr1279 | Nr1280
Nr1281 | Nr1282 | Nr1283 | Nr1284 | Nr1285 | Nr1286 | Nr1287 | Nr1288
Nr1289 | Nr1290 | Nr1291 | Nr1292 | Nr1293 | Nr1294 | Nr1295 | Nr1296
Nr1297 | Nr1298 | Nr1299 | Nr1300 | Nr1301 | Nr1302 | Nr1303 | Nr1304
Nr1305 | Nr1306 | Nr1307 | Nr1308 | Nr1309 | Nr1310 | Nr1311 | Nr1312
Nr1313 | Nr1314 | Nr1315 | Nr1316 | Nr1317 | Nr1318 | Nr1319 | Nr1320
Nr1321 | Nr1322 | Nr1323 | Nr1324 | Nr1325 | Nr1326 | Nr1327 | Nr1328
Nr1329 | Nr1330 | Nr1331 | Nr1332 | Nr1333 | Nr1334 | Nr1335 | Nr1336
Nr1337 | Nr1338 | Nr1339 | Nr1340 | Nr1341 | Nr1342 | Nr1343 | Nr1344
Nr1345 | Nr1346 | Nr1347 | Nr1348 | Nr1349 | Nr1350 | Nr1351 | Nr1352
Nr1353 | Nr1354 | Nr1355 | Nr1356 | Nr1357 | Nr1358 | Nr1359 | Nr1360
Nr1361 | Nr1362 | Nr1363 | Nr1364 | Nr1365 | Nr1366 | Nr1367 | Nr1368
Nr1369 | Nr1370 | Nr1371 | Nr1372 | Nr1373 | Nr1374 | Nr1375 | Nr1376
Nr1377 | Nr1378 | Nr1379 | Nr1380 | Nr1381 | Nr1382 | Nr1383 | Nr1384
Nr1385 | Nr1386 | Nr1387 | Nr1388 | Nr1389 | Nr1390 | Nr1391 | Nr1392
Nr1393 | Nr1394 | Nr1395 | Nr1396 | Nr1397 | Nr1398 | Nr1399 | Nr1400
Nr1401 | Nr1402 | Nr1403 | Nr1404 | Nr1405 | Nr1406 | Nr1407 | Nr1408
Nr1409 | Nr1410 | Nr1411 | Nr1412 | Nr1413 | Nr1414 | Nr1415 | Nr1416
Nr1417 | Nr1418 | Nr1419 | Nr1420 | Nr1421 | Nr1422 | Nr1423 | Nr1424
Nr1425 | Nr1426 | Nr1427 | Nr1428 | Nr1429 | Nr1430 | Nr1431 | Nr1432
Nr1433 | Nr1434 | Nr1435 | Nr1436 | Nr1437 | Nr1438 | Nr1439 | Nr1440
Nr1441 | Nr1442 | Nr1443 | Nr1444 | Nr1445 | Nr1446 | Nr1447 | Nr1448
Nr1449 | Nr1450 | Nr1451 | Nr1452 | Nr1453 | Nr1454 | Nr1455 | Nr1456
Nr1457 | Nr1458 | Nr1459 | Nr1460 | Nr1461 | Nr1462 | Nr1463 | Nr1464
Nr1465 | Nr1466 | Nr1467 | Nr1468 | Nr1469 | Nr1470 | Nr1471 | Nr1472
Nr1473 | Nr1474 | Nr1475 | Nr1476 | Nr1477 | Nr1478 | Nr1479 | Nr1480
Nr1481 | Nr1482 | Nr1483 | Nr1484 | Nr1485 | Nr1486 | Nr1487 | Nr1488
Nr1489 | Nr1490 | Nr1491 | Nr1492 | Nr1493 | Nr1494 | Nr1495 | Nr1496
Nr1497 | Nr1498 | Nr1499 | Nr1500 | Nr1501 | Nr1502 | Nr1503 | Nr1504
Nr1505 | Nr1506 | Nr1507 | Nr1508 | Nr1509 | Nr1510 | Nr1511 | Nr1512
Nr1513 | Nr1514 | Nr1515 | Nr1516 | Nr1517 | Nr1518 | Nr1519 | Nr1520
Nr1521 | Nr1522 | Nr1523 | Nr1524 | Nr1525 | Nr1526 | Nr1527 | Nr1528
Nr1529 | Nr1530 | Nr1531 | Nr1532 | Nr1533 | Nr1534 | Nr1535 | Nr1536
```

(continues on next page)

(continued from previous page)

```
Nr1537 | Nr1538 | Nr1539 | Nr1540 | Nr1541 | Nr1542 | Nr1543 | Nr1544
Nr1545 | Nr1546 | Nr1547 | Nr1548 | Nr1549 | Nr1550 | Nr1551 | Nr1552
Nr1553 | Nr1554 | Nr1555 | Nr1556 | Nr1557 | Nr1558 | Nr1559 | Nr1560
Nr1561 | Nr1562 | Nr1563 | Nr1564 | Nr1565 | Nr1566 | Nr1567 | Nr1568
Nr1569 | Nr1570 | Nr1571 | Nr1572 | Nr1573 | Nr1574 | Nr1575 | Nr1576
Nr1577 | Nr1578 | Nr1579 | Nr1580 | Nr1581 | Nr1582 | Nr1583 | Nr1584
Nr1585 | Nr1586 | Nr1587 | Nr1588 | Nr1589 | Nr1590 | Nr1591 | Nr1592
Nr1593 | Nr1594 | Nr1595 | Nr1596 | Nr1597 | Nr1598 | Nr1599 | Nr1600
Nr1601 | Nr1602 | Nr1603 | Nr1604 | Nr1605 | Nr1606 | Nr1607 | Nr1608
Nr1609 | Nr1610 | Nr1611 | Nr1612 | Nr1613 | Nr1614 | Nr1615 | Nr1616
Nr1617 | Nr1618 | Nr1619 | Nr1620 | Nr1621 | Nr1622 | Nr1623 | Nr1624
Nr1625 | Nr1626 | Nr1627 | Nr1628 | Nr1629 | Nr1630 | Nr1631 | Nr1632
Nr1633 | Nr1634 | Nr1635 | Nr1636 | Nr1637 | Nr1638 | Nr1639 | Nr1640
Nr1641 | Nr1642 | Nr1643 | Nr1644 | Nr1645 | Nr1646 | Nr1647 | Nr1648
Nr1649 | Nr1650 | Nr1651 | Nr1652 | Nr1653 | Nr1654 | Nr1655 | Nr1656
Nr1657 | Nr1658 | Nr1659 | Nr1660 | Nr1661 | Nr1662 | Nr1663 | Nr1664
Nr1665 | Nr1666 | Nr1667 | Nr1668 | Nr1669 | Nr1670 | Nr1671 | Nr1672
Nr1673 | Nr1674 | Nr1675 | Nr1676 | Nr1677 | Nr1678 | Nr1679 | Nr1680
Nr1681 | Nr1682 | Nr1683 | Nr1684 | Nr1685 | Nr1686 | Nr1687 | Nr1688
Nr1689 | Nr1690 | Nr1691 | Nr1692 | Nr1693 | Nr1694 | Nr1695 | Nr1696
Nr1697 | Nr1698 | Nr1699 | Nr1700 | Nr1701 | Nr1702 | Nr1703 | Nr1704
Nr1705 | Nr1706 | Nr1707 | Nr1708 | Nr1709 | Nr1710 | Nr1711 | Nr1712
Nr1713 | Nr1714 | Nr1715 | Nr1716 | Nr1717 | Nr1718 | Nr1719 | Nr1720
Nr1721 | Nr1722 | Nr1723 | Nr1724 | Nr1725 | Nr1726 | Nr1727 | Nr1728
Nr1729 | Nr1730 | Nr1731 | Nr1732 | Nr1733 | Nr1734 | Nr1735 | Nr1736
Nr1737 | Nr1738 | Nr1739 | Nr1740 | Nr1741 | Nr1742 | Nr1743 | Nr1744
Nr1745 | Nr1746 | Nr1747 | Nr1748 | Nr1749 | Nr1750 | Nr1751 | Nr1752
Nr1753 | Nr1754 | Nr1755 | Nr1756 | Nr1757 | Nr1758 | Nr1759 | Nr1760
Nr1761 | Nr1762 | Nr1763 | Nr1764 | Nr1765 | Nr1766 | Nr1767 | Nr1768
Nr1769 | Nr1770 | Nr1771 | Nr1772 | Nr1773 | Nr1774 | Nr1775 | Nr1776
Nr1777 | Nr1778 | Nr1779 | Nr1780 | Nr1781 | Nr1782 | Nr1783 | Nr1784
Nr1785 | Nr1786 | Nr1787 | Nr1788 | Nr1789 | Nr1790 | Nr1791 | Nr1792
Nr1793 | Nr1794 | Nr1795 | Nr1796 | Nr1797 | Nr1798 | Nr1799 | Nr1800
Nr1801 | Nr1802 | Nr1803 | Nr1804 | Nr1805 | Nr1806 | Nr1807 | Nr1808
Nr1809 | Nr1810 | Nr1811 | Nr1812 | Nr1813 | Nr1814 | Nr1815 | Nr1816
Nr1817 | Nr1818 | Nr1819 | Nr1820 | Nr1821 | Nr1822 | Nr1823 | Nr1824
Nr1825 | Nr1826 | Nr1827 | Nr1828 | Nr1829 | Nr1830 | Nr1831 | Nr1832
Nr1833 | Nr1834 | Nr1835 | Nr1836 | Nr1837 | Nr1838 | Nr1839 | Nr1840
Nr1841 | Nr1842 | Nr1843 | Nr1844 | Nr1845 | Nr1846 | Nr1847 | Nr1848
Nr1849 | Nr1850 | Nr1851 | Nr1852 | Nr1853 | Nr1854 | Nr1855 | Nr1856
Nr1857 | Nr1858 | Nr1859 | Nr1860 | Nr1861 | Nr1862 | Nr1863 | Nr1864
Nr1865 | Nr1866 | Nr1867 | Nr1868 | Nr1869 | Nr1870 | Nr1871 | Nr1872
Nr1873 | Nr1874 | Nr1875 | Nr1876 | Nr1877 | Nr1878 | Nr1879 | Nr1880
Nr1881 | Nr1882 | Nr1883 | Nr1884 | Nr1885 | Nr1886 | Nr1887 | Nr1888
Nr1889 | Nr1890 | Nr1891 | Nr1892 | Nr1893 | Nr1894 | Nr1895 | Nr1896
Nr1897 | Nr1898 | Nr1899 | Nr1900 | Nr1901 | Nr1902 | Nr1903 | Nr1904
Nr1905 | Nr1906 | Nr1907 | Nr1908 | Nr1909 | Nr1910 | Nr1911 | Nr1912
Nr1913 | Nr1914 | Nr1915 | Nr1916 | Nr1917 | Nr1918 | Nr1919 | Nr1920
Nr1921 | Nr1922 | Nr1923 | Nr1924 | Nr1925 | Nr1926 | Nr1927 | Nr1928
Nr1929 | Nr1930 | Nr1931 | Nr1932 | Nr1933 | Nr1934 | Nr1935 | Nr1936
Nr1937 | Nr1938 | Nr1939 | Nr1940 | Nr1941 | Nr1942 | Nr1943 | Nr1944
Nr1945 | Nr1946 | Nr1947 | Nr1948 | Nr1949 | Nr1950 | Nr1951 | Nr1952
```

(continues on next page)

(continued from previous page)

```
Nr1953 | Nr1954 | Nr1955 | Nr1956 | Nr1957 | Nr1958 | Nr1959 | Nr1960
Nr1961 | Nr1962 | Nr1963 | Nr1964 | Nr1965 | Nr1966 | Nr1967 | Nr1968
Nr1969 | Nr1970 | Nr1971 | Nr1972 | Nr1973 | Nr1974 | Nr1975 | Nr1976
Nr1977 | Nr1978 | Nr1979 | Nr1980 | Nr1981 | Nr1982 | Nr1983 | Nr1984
Nr1985 | Nr1986 | Nr1987 | Nr1988 | Nr1989 | Nr1990 | Nr1991 | Nr1992
Nr1993 | Nr1994 | Nr1995 | Nr1996 | Nr1997 | Nr1998 | Nr1999 | Nr2000
```

CHAPTER

FIVE

EXAMPLES

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

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

from RsCmwBase import *  # install from pypi.org

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

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

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

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

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

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


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

(continues on next page)

(continued from previous page)

```
↪{event_args.message}')


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

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

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

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

CHAPTER

SIX

RSCMPX_NIOTMEAS API STRUCTURE

Global RepCaps

```
driver = RsCMPX_NiotMeas('TCPIP::192.168.2.101::hislip0')
# Instance range: Inst1 .. Inst16
rc = driver.repcap_instance_get()
driver.repcap_instance_set(repcap.Instance.Inst1)
```

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

310 total commands, 4 Subgroups, 0 group commands

Initializes new RsCMPX_NiotMeas session.

Parameter options tokens examples:

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

- `ViClearExeMode = Disabled` - viClear() execution mode. Default: `execute_on_all`
- `OpcQueryAfterWrite = True` - same as driver.utilities.opc_query_after_write = True. Default: `False`
- `StbInErrorCheck = False` - if true, the driver checks errors with *STB? If false, it uses SYST:ERR?. Default: `True`
- `ScpiQuotes = double'. - for SCPI commands, you can define how strings are quoted. With single or double quotes. Possible values: single | double | {char}. Default: ``single`
- `LoggingMode = On` - Sets the logging status right from the start. Default: `Off`
- `LoggingName = 'MyDevice'` - Sets the name to represent the session in the log entries. Default: `'resource_name'`
- `LogToGlobalTarget = True` - Sets the logging target to the class-property previously set with RsCMPX_NiotMeas.set_global_logging_target() Default: `False`
- `LoggingToConsole = True` - Immediately starts logging to the console. Default: False
- `LoggingToUdp = True` - Immediately starts logging to the UDP port. Default: False
- `LoggingUdpPort = 49200` - UDP port to log to. Default: 49200

Parameters

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

static assert_minimum_version(*min_version: str*) → None

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

classmethod clear_global_logging_relative_timestamp() → None

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

close() → None

Closes the active RsCMPX_NiotMeas session.

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

Creates a new RsCMPX_NiotMeas 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 Configure

class ConfigureCls

Configure commands group definition. 95 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1 NiotMeas

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:BAND
```

class NiotMeasCls

NiotMeas commands group definition. 95 total commands, 3 Subgroups, 1 group commands

get_band() → Band

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:BAND
value: enums.Band = driver.configure.niotMeas.get_band()
```

Selects the operating band (OB) .

return
band: No help available

set_band(*band: Band*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:BAND
driver.configure.niotMeas.set_band(band = enums.Band.OB1)
```

Selects the operating band (OB) .

param band
No help available

Cloning the Group

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

Subgroups

6.1.1.1 MultiEval

SCPI Commands :

```
CONFigure:NIOT:MEASurement<instance>:MEValuation:FSYRange
CONFigure:NIOT:MEASurement<Instance>:MEValuation:TOUT
CONFigure:NIOT:MEASurement<Instance>:MEValuation:DMODe
CONFigure:NIOT:MEASurement<Instance>:MEValuation:MMODe
CONFigure:NIOT:MEASurement<Instance>:MEValuation:REPetition
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCONdition
CONFigure:NIOT:MEASurement<Instance>:MEValuation:MOEXception
CONFigure:NIOT:MEASurement<Instance>:MEValuation:CPRefix
CONFigure:NIOT:MEASurement<Instance>:MEValuation:CBANdwidth
CONFigure:NIOT:MEASurement<Instance>:MEValuation:PLCid
CONFigure:NIOT:MEASurement<Instance>:MEValuation:DSS
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCSPacing
CONFigure:NIOT:MEASurement<Instance>:MEValuation:NPFormat
CONFigure:NIOT:MEASurement<Instance>:MEValuation:NREPetitions
CONFigure:NIOT:MEASurement<Instance>:MEValuation:NRUNits
CONFigure:NIOT:MEASurement<Instance>:MEValuation:NSLots
```

class MultiEvalCls

MultiEval commands group definition. 62 total commands, 8 Subgroups, 16 group commands

get_channel_bw() → ChannelBw

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:CBANdwidth
value: enums.ChannelBw = driver.configure.niotMeas.multiEval.get_channel_bw()
```

No command help available

return
channel_bw: No help available

get_cprefix() → CyclicPrefix

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:CPRefix
value: enums.CyclicPrefix = driver.configure.niotMeas.multiEval.get_cprefix()
```

No command help available

return
cyclic_prefix: No help available

get_dmode() → Mode

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:DMODe
value: enums.Mode = driver.configure.niotMeas.multiEval.get_dmode()
```

No command help available

return
mode: No help available

get_dss() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:DSS
value: int = driver.configure.niotMeas.multiEval.get_dss()
```

Specifies the delta sequence shift value (ss) used to calculate the sequence shift pattern for the NPUSCH.

return
delta_seq_shift: No help available

get_fsy_range() → LowHigh

```
# SCPI: CONFigure:NIOT:MEASurement<instance>:MEValuation:FSYRange
value: enums.LowHigh = driver.configure.niotMeas.multiEval.get_fsy_range()
```

Specifies the frequency synchronization range.

return
fsr: No help available

get_mmode() → MeasurementMode

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:MMODe
value: enums.MeasurementMode = driver.configure.niotMeas.multiEval.get_mmode()
```

Selects the measurement mode.

return
measurement_mode: NORMal: normal mode MELMode: multi-evaluation list mode

get_mo_exception() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:MOEXception
value: bool = driver.configure.niotMeas.multiEval.get_mo_exception()
```

Specifies whether measurement results that the CMP180 identifies as faulty or inaccurate are rejected.

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

get_np_format() → NpuschFormat

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NPFormat
value: enums.NpuschFormat = driver.configure.niotMeas.multiEval.get_np_format()
```

Specifies the format of the NPUSCH.

return
npusch_format: No help available

get_nr_units() → NofRsrcUnits

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NRUNits
value: enums.NofRsrcUnits = driver.configure.niotMeas.multiEval.get_nr_units()
```

Specifies the number of resource units allocated for the NPUSCH.

return
nof_ru: No help available

get_nrepetitions() → NofRepetitions

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NREPetitions
value: enums.NofRepetitions = driver.configure.niotMeas.multiEval.get_
↪nrepetitions()
```

Specifies the number of NPUSCH repetitions.

return
nof_repetitions: Number of NPUSCH repetitions: 1, 2, 4, ..., 512, 1024, 2048

get_nslots() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NSLots
value: int = driver.configure.niotMeas.multiEval.get_nslots()
```

Configures the length of the captured sequence of slots.

return
nof_slots: No help available

get_plc_id() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:PLCid
value: int = driver.configure.niotMeas.multiEval.get_plc_id()
```

Specifies the physical layer cell ID.

return
phs_layer_cell_id: No help available

get_repetition() → Repeat

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:REPetition
value: enums.Repeat = driver.configure.niotMeas.multiEval.get_repetition()
```

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

return
repetition: SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

get_sc_spacing() → SubCarrSpacing

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCSPacing
value: enums.SubCarrSpacing = driver.configure.niotMeas.multiEval.get_sc_
↪spacing()
```

Selects the subcarrier spacing.

return
sub_carr_spacing: 3.75 kHz or 15 kHz

get_scondition() → StopCondition

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCONdition
value: enums.StopCondition = driver.configure.niotMeas.multiEval.get_
↪scondition()
```

Qualifies whether the measurement is stopped after a failed limit check or continued. SLFail means that the measurement is stopped and reaches the RDY state when one of the results exceeds the limits.

return
stop_condition: NONE: Continue measurement irrespective of the limit check SLFail: Stop measurement on limit failure

get_timeout() → float

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:TOUT
value: float = driver.configure.niotMeas.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 manually. When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return
timeout: No help available

set_channel_bw(*channel_bw: ChannelBw*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:CBANdwidth
driver.configure.niotMeas.multiEval.set_channel_bw(channel_bw = enums.ChannelBw.
↪B200)
```

No command help available

param channel_bw
No help available

set_cprefix(*cyclic_prefix: CyclicPrefix*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:CPRefix
driver.configure.niotMeas.multiEval.set_cprefix(cyclic_prefix = enums.
↪CyclicPrefix.EXTended)
```

No command help available

param cyclic_prefix
No help available

set_dmode(*mode: Mode*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:DMODe
driver.configure.niotMeas.multiEval.set_dmode(mode = enums.Mode.FDD)
```

No command help available

param mode
No help available

set_dss(*delta_seq_shift: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:DSS
driver.configure.niotMeas.multiEval.set_dss(delta_seq_shift = 1)
```

Specifies the delta sequence shift value (ss) used to calculate the sequence shift pattern for the NPUSCH.

param delta_seq_shift
No help available

set_fsy_range(*fsr: LowHigh*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<instance>:MEValuation:FSYRange
driver.configure.niotMeas.multiEval.set_fsy_range(fsr = enums.LowHigh.HIGH)
```

Specifies the frequency synchronization range.

param fsr
No help available

set_mmode(*measurement_mode: MeasurementMode*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:MMODe
driver.configure.niotMeas.multiEval.set_mmode(measurement_mode = enums.
↪MeasurementMode.MELMode)
```

Selects the measurement mode.

param measurement_mode
NORMal: normal mode MELMode: multi-evaluation list mode

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:MOEXception
driver.configure.niotMeas.multiEval.set_mo_exception(meas_on_exception = False)
```

Specifies whether measurement results that the CMP180 identifies as faulty or inaccurate are rejected.

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

set_np_format(*npusch_format: NpuschFormat*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NPFormat
driver.configure.niotMeas.multiEval.set_np_format(npusch_format = enums.
↪NpuschFormat.F1)
```

Specifies the format of the NPUSCH.

param npusch_format
No help available

set_nr_units(*nof_ru: NofRsrcUnits*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NRUNits
driver.configure.niotMeas.multiEval.set_nr_units(nof_ru = enums.NofRsrcUnits.
↪NRU01)
```

Specifies the number of resource units allocated for the NPUSCH.

param nof_ru
No help available

set_nrepetitions(*nof_repetitions: NofRepetitions*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NREPetitions
driver.configure.niotMeas.multiEval.set_nrepetitions(nof_repetitions = enums.
↪NofRepetitions.NR1)
```

Specifies the number of NPUSCH repetitions.

param nof_repetitions
Number of NPUSCH repetitions: 1, 2, 4, ..., 512, 1024, 2048

set_nslots(*nof_slots: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:NSLots
driver.configure.niotMeas.multiEval.set_nslots(nof_slots = 1)
```

Configures the length of the captured sequence of slots.

param nof_slots
No help available

set_plc_id(*phs_layer_cell_id: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:PLCid
driver.configure.niotMeas.multiEval.set_plc_id(phs_layer_cell_id = 1)
```

Specifies the physical layer cell ID.

param phs_layer_cell_id
No help available

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:REPetition
driver.configure.niotMeas.multiEval.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

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

param repetition
SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

set_sc_spacing(*sub_carr_spacing: SubCarrSpacing*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCSPacing
driver.configure.niotMeas.multiEval.set_sc_spacing(sub_carr_spacing = enums.
↪SubCarrSpacing.S15K)
```

Selects the subcarrier spacing.

param sub_carr_spacing
3.75 kHz or 15 kHz

set_scondition(*stop_condition: StopCondition*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCONdition
driver.configure.niotMeas.multiEval.set_scondition(stop_condition = enums.
↪StopCondition.NONE)
```

Qualifies whether the measurement is stopped after a failed limit check or continued. SLFail means that the measurement is stopped and reaches the RDY state when one of the results exceeds the limits.

param stop_condition
NONE: Continue measurement irrespective of the limit check SLFail: Stop measurement on limit failure

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:TOUT
driver.configure.niotMeas.multiEval.set_timeout(timeout = 1.0)
```

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

param timeout
No help available

Cloning the Group

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

Subgroups

6.1.1.1.1 Limit<Limit>

RepCap Settings

```
# Range: Nr1 .. Nr12
rc = driver.configure.niotMeas.multiEval.limit.repcap_limit_get()
driver.configure.niotMeas.multiEval.limit.repcap_limit_set(repcap.Limit.Nr1)
```

class LimitCls

Limit commands group definition. 12 total commands, 9 Subgroups, 0 group commands Repeated Capability: Limit, default value after init: Limit.Nr1

Cloning the Group

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

Subgroups

6.1.1.1.1.1 Aclr

class AclrCls

Aclr 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.niotMeas.multiEval.limit.aclr.clone()
```

Subgroups

6.1.1.1.1.2 Gsm

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:ACLR:GSM
```

class GsmCls

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

class GsmStruct

Response structure. Fields:

- Relative_Level: float or bool: No parameter help available
- Absolute_Level: float or bool: No parameter help available

get() → GsmStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:ACLR:GSM
value: GsmStruct = driver.configure.niotMeas.multiEval.limit.aclr.gsm.get()
```

Defines a relative and absolute limit for the ACLR measured in the adjacent GSM channel.

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

set(*relative_level: float*, *absolute_level: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:ACLR:GSM
driver.configure.niotMeas.multiEval.limit.aclr.gsm.set(relative_level = 1.0,␣
↪absolute_level = 1.0)
```

Defines a relative and absolute limit for the ACLR measured in the adjacent GSM channel.

param relative_level
(float or boolean) No help available

param absolute_level
(float or boolean) No help available

6.1.1.1.1.3 Utra

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:ACLR:UTRA
```

class UtraCls

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

class UtraStruct

Response structure. Fields:

- Relative_Level: float or bool: No parameter help available
- Absolute_Level: float or bool: No parameter help available

get() → UtraStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:ACLR:UTRA
value: UtraStruct = driver.configure.niotMeas.multiEval.limit.aclr.utra.get()
```

Defines a relative and absolute limit for the ACLR measured in the adjacent UTRA channel.

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

set(*relative_level: float*, *absolute_level: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:ACLR:UTRA
driver.configure.niotMeas.multiEval.limit.aclr.utra.set(relative_level = 1.0,␣
↪absolute_level = 1.0)
```

Defines a relative and absolute limit for the ACLR measured in the adjacent UTRA channel.

param relative_level
(float or boolean) No help available

param absolute_level
(float or boolean) No help available

6.1.1.1.1.4 EvMagnitude

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:EVMagnitude
```

class EvMagnitudeCls

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

class EvMagnitudeStruct

Response structure. Fields:

- Rms: float or bool: No parameter help available
- Peak: float or bool: No parameter help available

get() → EvMagnitudeStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:EVMagnitude
value: EvMagnitudeStruct = driver.configure.niotMeas.multiEval.limit.
↪evMagnitude.get()
```

Defines upper limits for the RMS and peak values of the error vector magnitude (EVM) .

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

set(*rms: float*, *peak: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:EVMagnitude
driver.configure.niotMeas.multiEval.limit.evMagnitude.set(rms = 1.0, peak = 1.0)
```

Defines upper limits for the RMS and peak values of the error vector magnitude (EVM) .

param rms
(float or boolean) No help available

param peak
(float or boolean) No help available

6.1.1.1.1.5 FreqError

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:FERRor
```

class FreqErrorCls

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

class FreqErrorStruct

Response structure. Fields:

- Freq_Err_Enable: bool: OFF: disables the limit check ON: enables the limit check
- Freq_Err_Low: float or bool: Upper limit for frequencies up to 1 GHz
- Freq_Err_High: float or bool: Upper limit for frequencies above 1 GHz

get() → FreqErrorStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:FERRor
value: FreqErrorStruct = driver.configure.niotMeas.multiEval.limit.freqError.
↪get()
```

Defines upper limits for the carrier frequency error.

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

set(*freq_err_enable: bool*, *freq_err_low: float*, *freq_err_high: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:FERRor
driver.configure.niotMeas.multiEval.limit.freqError.set(freq_err_enable = False,
↪ freq_err_low = 1.0, freq_err_high = 1.0)
```

Defines upper limits for the carrier frequency error.

param freq_err_enable
OFF: disables the limit check ON: enables the limit check

param freq_err_low
(float or boolean) Upper limit for frequencies up to 1 GHz

param freq_err_high
(float or boolean) Upper limit for frequencies above 1 GHz

6.1.1.1.1.6 Ibe

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IBE
```

class IbeCls

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

class IbeStruct

Response structure. Fields:

- Enable: bool: OFF: disables the limit check ON: enables the limit check
- Minimum: float: No parameter help available
- Sc_Power: float: No parameter help available
- Iq_Image: float: No parameter help available

get() → IbeStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IBE
value: IbeStruct = driver.configure.niotMeas.multiEval.limit.ibe.get()
```

Defines parameters used for calculation of an upper limit for the inband emissions, see 'Modulation limits: inband emissions'.

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

set(*enable: bool*, *minimum: float*, *sc_power: float*, *iq_image: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IBE
driver.configure.niotMeas.multiEval.limit.ibe.set(enable = False, minimum = 1.0,
↪ sc_power = 1.0, iq_image = 1.0)
```

Defines parameters used for calculation of an upper limit for the inband emissions, see 'Modulation limits: inband emissions'.

param enable
OFF: disables the limit check ON: enables the limit check

param minimum
No help available

param sc_power
No help available

param iq_image
No help available

Cloning the Group

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

Subgroups

6.1.1.1.1.7 IqOffset

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IBE:IQOFfset
```

class IqOffsetCls

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

class IqOffsetStruct

Response structure. Fields:

- Offset_1: float: Offset for high TX power range
- Offset_2: float: Offset for intermediate TX power range
- Offset_3: float: Offset for low TX power range

get() → IqOffsetStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IBE:IQOFfset
value: IqOffsetStruct = driver.configure.niotMeas.multiEval.limit.ibe.iqOffset.
↪get()
```

Defines I/Q origin offset values used for calculation of an upper limit for the inband emissions. Three different values can be set for three TX power ranges, see 'Modulation limits: inband emissions'.

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

set(*offset_1: float*, *offset_2: float*, *offset_3: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IBE:IQOFfset
driver.configure.niotMeas.multiEval.limit.ibe.iqOffset.set(offset_1 = 1.0,␣
↪offset_2 = 1.0, offset_3 = 1.0)
```

Defines I/Q origin offset values used for calculation of an upper limit for the inband emissions. Three different values can be set for three TX power ranges, see 'Modulation limits: inband emissions'.

param offset_1
Offset for high TX power range

param offset_2
Offset for intermediate TX power range

param offset_3
Offset for low TX power range

6.1.1.1.1.8 IqOffset

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IQOFfset
```

class IqOffsetCls

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

class IqOffsetStruct

Response structure. Fields:

- Enable: bool: OFF: disables the limit check ON: enables the limit check
- Offset_1: float: I/Q origin offset limit for high TX power range
- Offset_2: float: I/Q origin offset limit for intermediate TX power range
- Offset_3: float: I/Q origin offset limit for low TX power range

get() → IqOffsetStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IQOFfset
value: IqOffsetStruct = driver.configure.niotMeas.multiEval.limit.iqOffset.get()
```

Defines upper limits for the I/Q origin offset. Three different limits can be set for three TX power ranges. For details, see 'Modulation limits: I/Q offset'.

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

set(*enable: bool*, *offset_1: float*, *offset_2: float*, *offset_3: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:IQOFfset
driver.configure.niotMeas.multiEval.limit.iqOffset.set(enable = False, offset_1␣
↪= 1.0, offset_2 = 1.0, offset_3 = 1.0)
```

Defines upper limits for the I/Q origin offset. Three different limits can be set for three TX power ranges. For details, see 'Modulation limits: I/Q offset'.

param enable
OFF: disables the limit check ON: enables the limit check

param offset_1
I/Q origin offset limit for high TX power range

param offset_2
I/Q origin offset limit for intermediate TX power range

param offset_3
I/Q origin offset limit for low TX power range

6.1.1.1.1.9 Merror

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:MERRor
```

class MerrorCls

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

class MerrorStruct

Response structure. Fields:

- Rms: float or bool: No parameter help available
- Peak: float or bool: No parameter help available

get() → MerrorStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:MERRor
value: MerrorStruct = driver.configure.niotMeas.multiEval.limit.merror.get()
```

Defines upper limits for the RMS and peak values of the magnitude error.

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

set(*rms: float*, *peak: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:MERRor
driver.configure.niotMeas.multiEval.limit.merror.set(rms = 1.0, peak = 1.0)
```

Defines upper limits for the RMS and peak values of the magnitude error.

param rms
(float or boolean) No help available

param peak
(float or boolean) No help available

6.1.1.1.1.10 Pdynamics

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:PDYNamics
```

class PdynamicsCls

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

class PdynamicsStruct

Response structure. Fields:

- Enable: bool: OFF: disables the limit check ON: enables the limit check
- On_Power_Upper: float: Upper limit for the 'ON power'
- On_Power_Lower: float: Lower limit for the 'ON power'
- Off_Power_Upper: float: Upper limit for the 'OFF power'

get() → PdynamicsStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:PDYNamics
value: PdynamicsStruct = driver.configure.niotMeas.multiEval.limit.pdynamics.
↪get()
```

Defines limits for the ON power and OFF power determined with the power dynamics measurement.

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

set(*enable: bool*, *on_power_upper: float*, *on_power_lower: float*, *off_power_upper: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:PDYNamics
driver.configure.niotMeas.multiEval.limit.pdynamics.set(enable = False, on_
↪power_upper = 1.0, on_power_lower = 1.0, off_power_upper = 1.0)
```

Defines limits for the ON power and OFF power determined with the power dynamics measurement.

param enable
OFF: disables the limit check ON: enables the limit check

param on_power_upper
Upper limit for the 'ON power'

param on_power_lower
Lower limit for the 'ON power'

param off_power_upper
Upper limit for the 'OFF power'

6.1.1.1.1.11 Perror

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:PERRor
```

class PerrorCls

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

class PerrorStruct

Response structure. Fields:

- Rms: float or bool: No parameter help available
- Peak: float or bool: No parameter help available

get() → PerrorStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:PERRor
value: PerrorStruct = driver.configure.niotMeas.multiEval.limit.perror.get()
```

Defines symmetric limits for the RMS and peak values of the phase error. The limit check fails if the absolute value of the measured phase error exceeds the specified values.

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

set(*rms: float*, *peak: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:PERRor
driver.configure.niotMeas.multiEval.limit.perror.set(rms = 1.0, peak = 1.0)
```

Defines symmetric limits for the RMS and peak values of the phase error. The limit check fails if the absolute value of the measured phase error exceeds the specified values.

param rms
(float or boolean) No help available

param peak
(float or boolean) No help available

6.1.1.1.1.12 SeMask

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:SEMask:OBWLimit
```

class SeMaskCls

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

get_obw_limit() → float

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:SEMask:OBWLimit
value: float or bool = driver.configure.niotMeas.multiEval.limit.seMask.get_obw_
↪limit()
```

Defines an upper limit for the occupied bandwidth.

return
obw_limit: (float or boolean) No help available

set_obw_limit(*obw_limit: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:SEMask:OBWLimit
driver.configure.niotMeas.multiEval.limit.seMask.set_obw_limit(obw_limit = 1.0)
```

Defines an upper limit for the occupied bandwidth.

param obw_limit
(float or boolean) No help available

Cloning the Group

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

Subgroups

6.1.1.1.1.13 Limit

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:SEMask:LIMit<nr>
```

class LimitCls

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

class LimitStruct

Response structure. Fields:

- Enable: bool: OFF: disables the check of these requirements ON: enables the check of these requirements
- Frequency_Start: float: Start frequency of the area, relative to the edges of the channel bandwidth.
- Frequency_Stop: float: Stop frequency of the area, relative to the edges of the channel bandwidth.
- Power_Level_Start: float: Upper limit at the FrequencyStart
- Power_Level_Stop: float: Upper limit at the FrequencyEnd

get(*limit=Limit.Default*) → LimitStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:SEMask:LIMit<nr>
value: LimitStruct = driver.configure.niotMeas.multiEval.limit.seMask.limit.
↪get(limit = repcap.Limit.Default)
```

Defines the emission mask area <no>.

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

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

set(*enable: bool*, *frequency_start: float*, *frequency_stop: float*, *power_level_start: float*, *power_level_stop: float*, *limit=Limit.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIMit:SEMask:LIMit<nr>
driver.configure.niotMeas.multiEval.limit.seMask.limit.set(enable = False,␣
↪frequency_start = 1.0, frequency_stop = 1.0, power_level_start = 1.0, power_
↪level_stop = 1.0, limit = repcap.Limit.Default)
```

Defines the emission mask area <no>.

param enable
OFF: disables the check of these requirements ON: enables the check of these requirements

param frequency_start
Start frequency of the area, relative to the edges of the channel bandwidth.

param frequency_stop
Stop frequency of the area, relative to the edges of the channel bandwidth.

param power_level_start
Upper limit at the FrequencyStart

param power_level_stop
Upper limit at the FrequencyEnd

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

6.1.1.1.2 ListPy

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:OSINdex
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:CMODe
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:NCONnections
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST
```

class ListPyCls

ListPy commands group definition. 12 total commands, 3 Subgroups, 4 group commands

get_cmode() → ParameterSetMode

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:CMODe
value: enums.ParameterSetMode = driver.configure.niotMeas.multiEval.listPy.get_
↪cmode()
```

Sets the connector mode, selecting whether all list mode segments use the same RF connection.

return
connector_mode: - GLOBal: Use the same RF connection for all segments, see ROUTe:NIOT:MEASi:SPATh. - LIST: Assign a connection to each segment, see CONFigure:NIOT:MEASi:MEValuation:LIST:SEGMentno:CIDX.

get_nconnections() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:NCONnections
value: int = driver.configure.niotMeas.multiEval.listPy.get_nconnections()
```

Sets the number of connections to be defined for the list mode, for connector mode LIST. Define the connections via ROUTe:NIOT:MEAS<i>:SPATh.

return
no_of_connections: The maximum number of connections is limited by the number of connectors per smart channel.

get_os_index() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:OSINdex
value: int or bool = driver.configure.niotMeas.multiEval.listPy.get_os_index()
```

Selects the segment to be displayed in offline mode. The index refers to the range of measured segments, see method RsCMPX_NiotMeas.Configure.NiotMeas.MultiEval.ListPy.Lrange.set. Setting a value also enables the offline mode.

return
offline_seg_index: (integer or boolean) No help available

get_value() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST
value: bool = driver.configure.niotMeas.multiEval.listPy.get_value()
```

Enables or disables the list mode.

return
enable: OFF: Disable list mode ON: Enable list mode

set_cmode(*connector_mode: ParameterSetMode*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:CMODe
driver.configure.niotMeas.multiEval.listPy.set_cmode(connector_mode = enums.
↪ParameterSetMode.GLOBal)
```

Sets the connector mode, selecting whether all list mode segments use the same RF connection.

param connector_mode

- GLOBal: Use the same RF connection for all segments, see ROUTe:NIOT:MEASi:SPATh.
- LIST: Assign a connection to each segment, see CONFigure:NIOT:MEASi:MEValuation:LIST:SEGMentno:CIDX.

set_nconnections(*no_of_connections: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:NCONnections
driver.configure.niotMeas.multiEval.listPy.set_nconnections(no_of_connections =␣
↪1)
```

Sets the number of connections to be defined for the list mode, for connector mode LIST. Define the connections via ROUTe:NIOT:MEAS<i>:SPATh.

param no_of_connections
The maximum number of connections is limited by the number of connectors per smart channel.

set_os_index(*offline_seg_index: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:OSINdex
driver.configure.niotMeas.multiEval.listPy.set_os_index(offline_seg_index = 1)
```

Selects the segment to be displayed in offline mode. The index refers to the range of measured segments, see method RsCMPX_NiotMeas.Configure.NiotMeas.MultiEval.ListPy.Lrange.set. Setting a value also enables the offline mode.

param offline_seg_index
(integer or boolean) No help available

set_value(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST
driver.configure.niotMeas.multiEval.listPy.set_value(enable = False)
```

Enables or disables the list mode.

param enable
OFF: Disable list mode ON: Enable list mode

Cloning the Group

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

Subgroups

6.1.1.1.2.1 Lrange

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:LRANge
```

class LrangeCls

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

class LrangeStruct

Response structure. Fields:

- Start_Index: int: First measured segment in the range of configured segments
- Nr_Segments: int: Number of measured segments

get() → LrangeStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:LRANge
value: LrangeStruct = driver.configure.niotMeas.multiEval.listPy.lrange.get()
```

Selects a range of measured segments. The segments must be configured using method RsCMPX_NiotMeas.Configure.NiotMeas. MultiEval.ListPy.Segment.Setup.set.

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

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

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:LRANge
driver.configure.niotMeas.multiEval.listPy.lrange.set(start_index = 1, nr_
↪segments = 1)
```

Selects a range of measured segments. The segments must be configured using method RsCMPX_NiotMeas.Configure.NiotMeas. MultiEval.ListPy.Segment.Setup.set.

param start_index
First measured segment in the range of configured segments

param nr_segments
Number of measured segments

6.1.1.1.2.2 Segment

RepCap Settings

```
# Range: Nr1 .. Nr2000
rc = driver.configure.niotMeas.multiEval.listPy.segment.repcap_segment_get()
driver.configure.niotMeas.multiEval.listPy.segment.repcap_segment_set(repcap.Segment.Nr1)
```

class SegmentCls

Segment commands group definition. 6 total commands, 6 Subgroups, 0 group commands Repeated Capability: Segment, default value after init: Segment.Nr1

Cloning the Group

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

Subgroups

6.1.1.1.2.3 Aclr

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:ACLR
```

class AclrCls

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

class AclrStruct

Response structure. Fields:

- Aclr_Statistics: int: Statistical length in slots
- **Aclr_Enable: bool: Enables or disables the measurement of ACLR results**
  - ON: ACLR results are measured according to the other enable flags in this command. ACLR results for which there is no explicit enable flag are also measured (e.g. power in the NB-IoT channel) .
  - OFF: No ACLR results at all are measured. The other enable flags in this command are ignored.
- Utra_Enable: bool: Enables or disables the evaluation of adjacent UTRA channels
- Gsm_Enable: bool: Enables or disables the evaluation of adjacent GSM channels

get(*segment=Segment.Default*) → AclrStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:ACLR
value: AclrStruct = driver.configure.niotMeas.multiEval.listPy.segment.aclr.
↪get(segment = repcap.Segment.Default)
```

Defines settings for ACLR measurements in list mode for segment <no>.

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

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

set(*aclr_statistics: int*, *aclr_enable: bool*, *utra_enable: bool*, *gsm_enable: bool*, *segment=Segment.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:ACLR
driver.configure.niotMeas.multiEval.listPy.segment.aclr.set(aclr_statistics = 1,
↪ aclr_enable = False, utra_enable = False, gsm_enable = False, segment =␣
↪repcap.Segment.Default)
```

Defines settings for ACLR measurements in list mode for segment <no>.

param aclr_statistics
Statistical length in slots

param aclr_enable
Enables or disables the measurement of ACLR results - ON: ACLR results are measured according to the other enable flags in this command. ACLR results for which there is no explicit enable flag are also measured (e.g. power in the NB-IoT channel) . - OFF: No ACLR results at all are measured. The other enable flags in this command are ignored.

param utra_enable
Enables or disables the evaluation of adjacent UTRA channels

param gsm_enable
Enables or disables the evaluation of adjacent GSM channels

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

6.1.1.1.2.4 Cidx

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:CIDX
```

class CidxCls

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

get(*segment=Segment.Default*) → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:CIDX
value: int = driver.configure.niotMeas.multiEval.listPy.segment.cidx.
↪get(segment = repcap.Segment.Default)
```

Selects the RF connection index for segment <no>. For definition of the connection indices, see ROUTe:NIOT:MEAS<i>:SPATh.

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

return
connection_index: Index of the connection to be used for the segment.

set(*connection_index: int*, *segment=Segment.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:CIDX
driver.configure.niotMeas.multiEval.listPy.segment.cidx.set(connection_index =␣
↪1, segment = repcap.Segment.Default)
```

Selects the RF connection index for segment <no>. For definition of the connection indices, see ROUTe:NIOT:MEAS<i>:SPATh.

param connection_index
Index of the connection to be used for the segment.

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

6.1.1.1.2.5 Modulation

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:MODulation
```

class ModulationCls

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

class ModulationStruct

Structure for setting input parameters. Fields:

- Mod_Statistics: int: Statistical length in slots
- **Modenable: bool: Enables or disables the measurement of modulation results**

  - ON: Modulation results are measured according to the other enable flags in this command. Modulation results for which there is no explicit enable flag are also measured (e.g. I/Q offset, frequency error and timing error) .
  - OFF: No modulation results at all are measured. The other enable flags in this command are ignored.

- Evm_Enable: bool: Enables or disables the measurement of EVM
- Mag_Error_Enable: bool: Enables or disables the measurement of magnitude error
- Phase_Err_Enable: bool: Enables or disables the measurement of phase error
- Ib_Eenable: bool: Enables or disables the measurement of inband emissions
- Mod_Scheme: enums.ModScheme: Modulation scheme used by the NB-IoT uplink signal: BPSK, QPSK, 16QAM For 16QAM,

multiple subcarriers per RU must be in use (No. of SC 1) . See 'Resource unit allocation'

get(*segment=Segment.Default*) → ModulationStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:MODulation
value: ModulationStruct = driver.configure.niotMeas.multiEval.listPy.segment.
↪modulation.get(segment = repcap.Segment.Default)
```

Defines settings for modulation measurements in list mode for segment <no>.

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

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

set(*structure: ModulationStruct*, *segment=Segment.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:MODulation
structure = driver.configure.niotMeas.multiEval.listPy.segment.modulation.
↪ModulationStruct()
structure.Mod_Statistics: int = 1
structure.Modenable: bool = False
structure.Evm_Enable: bool = False
structure.Mag_Error_Enable: bool = False
structure.Phase_Err_Enable: bool = False
structure.Ib_Eenable: bool = False
structure.Mod_Scheme: enums.ModScheme = enums.ModScheme.BPSK
driver.configure.niotMeas.multiEval.listPy.segment.modulation.set(structure,␣
↪segment = repcap.Segment.Default)
```

Defines settings for modulation measurements in list mode for segment <no>.

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

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

6.1.1.1.2.6 SeMask

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask
```

class SeMaskCls

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

class SeMaskStruct

Response structure. Fields:

- Sem_Statistics: int: Statistical length in slots
- Sem_Enable: bool: Enables or disables the measurement of spectrum emission trace and margin results

get(*segment=Segment.Default*) → SeMaskStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask
value: SeMaskStruct = driver.configure.niotMeas.multiEval.listPy.segment.seMask.
↪get(segment = repcap.Segment.Default)
```

Defines settings for spectrum emission measurements in list mode for segment <no>.

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

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

set(*sem_statistics: int*, *sem_enable: bool*, *segment=Segment.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask
driver.configure.niotMeas.multiEval.listPy.segment.seMask.set(sem_statistics =␣
↪1, sem_enable = False, segment = repcap.Segment.Default)
```

Defines settings for spectrum emission measurements in list mode for segment <no>.

param sem_statistics
Statistical length in slots

param sem_enable
Enables or disables the measurement of spectrum emission trace and margin results

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

6.1.1.1.2.7 Setup

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SETup
```

class SetupCls

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

class SetupStruct

Structure for setting input parameters. Fields:

- Segment_Length: int: Number of slots in the segment
- Level: float: Expected nominal power in the segment. The range can be calculated as follows: Range (Expected Nominal Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the specifications document.
- Band: enums.Band: No parameter help available
- Frequency: float: Carrier center frequency used in the segment
- Npusch_Format: enums.NpuschFormat: Format of the NPUSCH.
- Nof_Subcarrier: int: Number of subcarriers per resource unit The allowed values have dependencies, see 'Resource unit allocation'.
- Start_Sc: int: Offset of the first allocated subcarrier from the edge of the transmission bandwidth For a subcarrier spacing of 3.75 kHz / 15 kHz, n equals 48 / 12.
- Nof_Ru_S: enums.NofRsrcUnits: Number of resource units allocated for the NPUSCH
- Nof_Repetitions: enums.NofRepetitionsList: Number of NPUSCH repetitions: 1, 2, 4, ..., 512, 1024, 2048
- **Retrigger_Flag: enums.RetriggerFlag: Specifies whether the measurement waits for a trigger event before measuring the segment, or not. For the first segment, the value OFF is always interpreted as ON. For subsequent segments, the retrigger flag is ignored for trigger mode ONCE and evaluated for trigger mode SEGMent, see [CMDLINKRESOLVED Trigger.NiotMeas.MultiEval.ListPy#Mode CMDLINKRESOLVED].**
  - OFF: Measure the segment without retrigger.
  - ON: Wait for a trigger event from the trigger source configured via TRIGger:NIOT:MEASi:MEValuation:SOURce.
  - IFPower: Wait for a trigger event from the trigger source IF Power.
- Evaluat_Offset: int: Number of slots at the beginning of the segment that are not evaluated

get(*segment=Segment.Default*) → SetupStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SETup
value: SetupStruct = driver.configure.niotMeas.multiEval.listPy.segment.setup.
↪get(segment = repcap.Segment.Default)
```

Defines the length and analyzer settings of segment <no>. This command must be sent for all segments to be measured (method RsCMPX_NiotMeas.Configure.NiotMeas.MultiEval.ListPy.Lrange.set) . For the supported frequency range, see 'Frequency ranges'.

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

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

set(*structure: SetupStruct*, *segment=Segment.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SETup
structure = driver.configure.niotMeas.multiEval.listPy.segment.setup.
↪SetupStruct()
structure.Segment_Length: int = 1
structure.Level: float = 1.0
structure.Band: enums.Band = enums.Band.OB1
structure.Frequency: float = 1.0
structure.Npusch_Format: enums.NpuschFormat = enums.NpuschFormat.F1
structure.Nof_Subcarrier: int = 1
structure.Start_Sc: int = 1
structure.Nof_Ru_S: enums.NofRsrcUnits = enums.NofRsrcUnits.NRU01
structure.Nof_Repetitions: enums.NofRepetitionsList = enums.NofRepetitionsList.
↪NR1
structure.Retrigger_Flag: enums.RetriggerFlag = enums.RetriggerFlag.IFPower
structure.Evaluat_Offset: int = 1
driver.configure.niotMeas.multiEval.listPy.segment.setup.set(structure, segment␣
↪= repcap.Segment.Default)
```

Defines the length and analyzer settings of segment <no>. This command must be sent for all segments to be measured (method RsCMPX_NiotMeas.Configure.NiotMeas.MultiEval.ListPy.Lrange.set) . For the supported frequency range, see 'Frequency ranges'.

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

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

6.1.1.1.2.8 SingleCmw

class SingleCmwCls

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

Cloning the Group

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

Subgroups

6.1.1.1.2.9 Connector

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:CMWS:CONNector
```

class ConnectorCls

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

get(*segment=Segment.Default*) → CmwsConnector

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:CMWS:CONNector
value: enums.CmwsConnector = driver.configure.niotMeas.multiEval.listPy.segment.
↪singleCmw.connector.get(segment = repcap.Segment.Default)
```

No command help available

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

return
cmws_connector: No help available

set(*cmws_connector: CmwsConnector*, *segment=Segment.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:CMWS:CONNector
driver.configure.niotMeas.multiEval.listPy.segment.singleCmw.connector.set(cmws_
↪connector = enums.CmwsConnector.R11, segment = repcap.Segment.Default)
```

No command help available

param cmws_connector
No help available

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

6.1.1.1.2.10 SingleCmw

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:CMWS:CMODe
```

class SingleCmwCls

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

get_cmode() → ParameterSetMode

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:CMWS:CMODe
value: enums.ParameterSetMode = driver.configure.niotMeas.multiEval.listPy.
↪singleCmw.get_cmode()
```

No command help available

return
connector_mode: No help available

set_cmode(*connector_mode: ParameterSetMode*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:LIST:CMWS:CMODe
driver.configure.niotMeas.multiEval.listPy.singleCmw.set_cmode(connector_mode =␣
↪enums.ParameterSetMode.GLOBal)
```

No command help available

param connector_mode
No help available

6.1.1.1.3 Modulation

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:MODulation:MSCHeme
```

class ModulationCls

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

get_mscheme() → ModScheme

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:MODulation:MSCHeme
value: enums.ModScheme = driver.configure.niotMeas.multiEval.modulation.get_
↪mscheme()
```

Selects the modulation scheme used by the measured signal.

return
mod_scheme: No help available

set_mscheme(*mod_scheme: ModScheme*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:MODulation:MSCHeme
driver.configure.niotMeas.multiEval.modulation.set_mscheme(mod_scheme = enums.
↪ModScheme.BPSK)
```

Selects the modulation scheme used by the measured signal.

param mod_scheme
No help available

Cloning the Group

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

Subgroups

6.1.1.1.3.1 EePeriods

class EePeriodsCls

EePeriods 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.niotMeas.multiEval.modulation.eePeriods.clone()
```

Subgroups

6.1.1.1.3.2 Npusch

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:MODulation:EEPeriods:NPUSch:LEADing
CONFigure:NIOT:MEASurement<Instance>:MEValuation:MODulation:EEPeriods:NPUSch:LAGGing
```

class NpuschCls

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

get_lagging() → LeadLag

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>
↪:MEValuation:MODulation:EEPeriods:NPUSch:LAGGing
value: enums.LeadLag = driver.configure.niotMeas.multiEval.modulation.eePeriods.
↪npusch.get_lagging()
```

Defines an exclusion period at the end of each NPUSCH transmission. The excluded symbols are ignored for modulation results.

return

lagging: OFF: no exclusion S1: exclude first symbol

get_leading() → LeadLag

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>
↪:MEValuation:MODulation:EEPeriods:NPUSch:LEADing
value: enums.LeadLag = driver.configure.niotMeas.multiEval.modulation.eePeriods.
↪npusch.get_leading()
```

Defines an exclusion period at the beginning of each NPUSCH transmission. The excluded symbols are ignored for modulation results.

return
leading: OFF: no exclusion S1: exclude first symbol

set_lagging(*lagging: LeadLag*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>
↪:MEValuation:MODulation:EEPeriods:NPUSch:LAGGing
driver.configure.niotMeas.multiEval.modulation.eePeriods.npusch.set_
↪lagging(lagging = enums.LeadLag.OFF)
```

Defines an exclusion period at the end of each NPUSCH transmission. The excluded symbols are ignored for modulation results.

param lagging
OFF: no exclusion S1: exclude first symbol

set_leading(*leading: LeadLag*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>
↪:MEValuation:MODulation:EEPeriods:NPUSch:LEADing
driver.configure.niotMeas.multiEval.modulation.eePeriods.npusch.set_
↪leading(leading = enums.LeadLag.OFF)
```

Defines an exclusion period at the beginning of each NPUSCH transmission. The excluded symbols are ignored for modulation results.

param leading
OFF: no exclusion S1: exclude first symbol

6.1.1.1.4 Pdynamics

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:TMASk
```

class PdynamicsCls

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

get_tmask() → TimeMask

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:TMASk
value: enums.TimeMask = driver.configure.niotMeas.multiEval.pdynamics.get_
↪tmask()
```

No command help available

return
time_mask: No help available

set_tmask(*time_mask: TimeMask*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:TMASk
driver.configure.niotMeas.multiEval.pdynamics.set_tmask(time_mask = enums.
↪TimeMask.GOO)
```

No command help available

param time_mask
No help available

6.1.1.1.5 Result

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult[:ALL]
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:EVMagnitude
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:MERRor
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PERRor
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:IEMissions
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:TXM
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:IQ
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:SEMask
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:ACLR
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PMONitor
CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PDYNamics
```

class ResultCls

Result commands group definition. 11 total commands, 0 Subgroups, 11 group commands

class AllStruct

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

- Evm: bool: Error vector magnitude OFF: Do not evaluate results. ON: Evaluate results.
- Magnitude_Error: bool: No parameter help available
- Phase_Error: bool: No parameter help available
- Inband_Emissions: bool: No parameter help available
- Iq: bool: I/Q constellation diagram
- Tx_Measurement: bool: TX measurement statistical overview
- Spec_Em_Mask: bool: Spectrum emission mask
- Aclr: bool: Adjacent channel leakage power ratio
- Power_Monitor: bool: No parameter help available
- Power_Dynamics: bool: No parameter help available

get_aclr() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:ACLR
value: bool = driver.configure.niotMeas.multiEval.result.get_aclr()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_all() → AllStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult[:ALL]
value: AllStruct = driver.configure.niotMeas.multiEval.result.get_all()
```

Enables or disables the evaluation of results in the multi-evaluation measurement. This command combines most other CONFigure:NIOT:MEAS<i>:MEValuation:RESult... commands.

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

get_ev_magnitude() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:EVMagnitude
value: bool = driver.configure.niotMeas.multiEval.result.get_ev_magnitude()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_iemissions() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:IEMissions
value: bool = driver.configure.niotMeas.multiEval.result.get_iemissions()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_iq() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:IQ
value: bool = driver.configure.niotMeas.multiEval.result.get_iq()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_merror() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:MERRor
value: bool = driver.configure.niotMeas.multiEval.result.get_merror()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_pdynamics() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PDYNamics
value: bool = driver.configure.niotMeas.multiEval.result.get_pdynamics()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor

- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

`get_perror()` → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PERRor
value: bool = driver.configure.niotMeas.multiEval.result.get_perror()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

`get_pmonitor()` → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PMONitor
value: bool = driver.configure.niotMeas.multiEval.result.get_pmonitor()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

`get_se_mask()` → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:SEMask
value: bool = driver.configure.niotMeas.multiEval.result.get_se_mask()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio

- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

`get_txm`() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:TXM
value: bool = driver.configure.niotMeas.multiEval.result.get_txm()
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

`set_aclr`(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:ACLR
driver.configure.niotMeas.multiEval.result.set_aclr(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

`set_all`(*value: AllStruct*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult[:ALL]
structure = driver.configure.niotMeas.multiEval.result.AllStruct()
structure.Evm: bool = False
structure.Magnitude_Error: bool = False
structure.Phase_Error: bool = False
structure.Inband_Emissions: bool = False
structure.Iq: bool = False
structure.Tx_Measurement: bool = False
```

(continues on next page)

(continued from previous page)

```
structure.Spec_Em_Mask: bool = False
structure.Aclr: bool = False
structure.Power_Monitor: bool = False
structure.Power_Dynamics: bool = False
driver.configure.niotMeas.multiEval.result.set_all(value = structure)
```

Enables or disables the evaluation of results in the multi-evaluation measurement. This command combines most other CONFigure:NIOT:MEAS<i>:MEValuation:RESult... commands.

param value
see the help for AllStruct structure arguments.

set_ev_magnitude(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:EVMagnitude
driver.configure.niotMeas.multiEval.result.set_ev_magnitude(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_iemissions(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:IEMissions
driver.configure.niotMeas.multiEval.result.set_iemissions(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_iq(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:IQ
driver.configure.niotMeas.multiEval.result.set_iq(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

`set_merror`(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:MERRor
driver.configure.niotMeas.multiEval.result.set_merror(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

`set_pdynamics`(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PDYNamics
driver.configure.niotMeas.multiEval.result.set_pdynamics(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_perror(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PERRor
driver.configure.niotMeas.multiEval.result.set_perror(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_pmonitor(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:PMONitor
driver.configure.niotMeas.multiEval.result.set_pmonitor(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_se_mask(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:SEMask
driver.configure.niotMeas.multiEval.result.set_se_mask(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

`set_txm`(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:RESult:TXM
driver.configure.niotMeas.multiEval.result.set_txm(enable = False)
```

Enables or disables the evaluation of results in the multi-evaluation measurement.
Table Header: Mnemonic / Square type / / Mnemonic / Square type

- EVMagnitude / Error vector magnitude / TXM / TX meas. statistical overview
- MERRor / Magnitude error / SEMask / Spectrum emission mask
- PERRor / Phase error / ACLR / Adj. channel leakage power ratio
- IEMissions / Inband emissions / PMONitor / Power monitor
- IQ / I/Q constellation diagram / PDYNamics / Power dynamics

param enable
OFF: Do not evaluate results. ON: Evaluate results.

6.1.1.1.6 Scount

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:MODulation
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:POWer
```

`class ScountCls`

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

`get_modulation`() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:MODulation
value: int = driver.configure.niotMeas.multiEval.scount.get_modulation()
```

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

return
statistic_count: Number of measurement intervals (slots)

`get_power`() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:POWer
value: int = driver.configure.niotMeas.multiEval.scount.get_power()
```

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

return
statistic_count: Number of measurement intervals

set_modulation(*statistic_count: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:MODulation
driver.configure.niotMeas.multiEval.scount.set_modulation(statistic_count = 1)
```

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

param statistic_count
Number of measurement intervals (slots)

set_power(*statistic_count: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:POWer
driver.configure.niotMeas.multiEval.scount.set_power(statistic_count = 1)
```

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

param statistic_count
Number of measurement intervals

Cloning the Group

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

Subgroups

6.1.1.1.6.1 Spectrum

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:SPECtrum:SEMask
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:SPECtrum:ACLR
```

class SpectrumCls

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

get_aclr() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:SPECtrum:ACLR
value: int = driver.configure.niotMeas.multiEval.scount.spectrum.get_aclr()
```

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

return
statistic_count: Number of measurement intervals (slots)

get_se_mask() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:SPECtrum:SEMask
value: int = driver.configure.niotMeas.multiEval.scount.spectrum.get_se_mask()
```

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

return
statistic_count: Number of measurement intervals (slots)

set_aclr(*statistic_count: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:SPECtrum:ACLR
driver.configure.niotMeas.multiEval.scount.spectrum.set_aclr(statistic_count =
↪1)
```

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

param statistic_count
Number of measurement intervals (slots)

set_se_mask(*statistic_count: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SCOunt:SPECtrum:SEMask
driver.configure.niotMeas.multiEval.scount.spectrum.set_se_mask(statistic_count
↪= 1)
```

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

param statistic_count
Number of measurement intervals (slots)

6.1.1.1.7 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.configure.niotMeas.multiEval.spectrum.clone()
```

Subgroups

6.1.1.1.7.1 Aclr

class AclrCls

Aclr 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.niotMeas.multiEval.spectrum.aclr.clone()
```

Subgroups

6.1.1.1.7.2 Enable

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SPECtrum:ACLR:ENABle
```

class EnableCls

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

class EnableStruct

Response structure. Fields:

- Utra: bool: No parameter help available
- Gsm: bool: No parameter help available

get() → EnableStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SPECtrum:ACLR:ENABle
value: EnableStruct = driver.configure.niotMeas.multiEval.spectrum.aclr.enable.
↪get()
```

Enables or disables the evaluation of the adjacent UTRA channels and the adjacent GSM channels.

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

set(*utra: bool*, *gsm: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SPECtrum:ACLR:ENABle
driver.configure.niotMeas.multiEval.spectrum.aclr.enable.set(utra = False, gsm␣
↪= False)
```

Enables or disables the evaluation of the adjacent UTRA channels and the adjacent GSM channels.

param utra
No help available

param gsm
No help available

6.1.1.1.7.3 SeMask

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SPECtrum:SEMask:OBWMode
```

class SeMaskCls

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

get_obw_mode() → ObwMode

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SPECtrum:SEMask:OBWMode
value: enums.ObwMode = driver.configure.niotMeas.multiEval.spectrum.seMask.get_
↪obw_mode()
```

No command help available

return
mode: No help available

set_obw_mode(*mode: ObwMode*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SPECtrum:SEMask:OBWMode
driver.configure.niotMeas.multiEval.spectrum.seMask.set_obw_mode(mode = enums.
↪ObwMode.BW99)
```

No command help available

param mode
No help available

6.1.1.1.8 Subcarrier

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:MEValuation:SUBCarrier
```

class SubcarrierCls

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

class SubcarrierStruct

Response structure. Fields:

- Nof_Subcarrier: int: Number of subcarriers per resource unit The allowed values have dependencies, see 'Resource unit allocation'.
- Offset: int: Offset of the first allocated subcarrier from the edge of the transmission bandwidth For a subcarrier spacing of 3.75 kHz / 15 kHz, n equals 48 / 12.

get() → SubcarrierStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SUBCarrier
value: SubcarrierStruct = driver.configure.niotMeas.multiEval.subcarrier.get()
```

Specifies the subcarrier configuration of the allocated resource units.

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

set(*nof_subcarrier: int*, *offset: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:MEValuation:SUBCarrier
driver.configure.niotMeas.multiEval.subcarrier.set(nof_subcarrier = 1, offset =␣
↪1)
```

Specifies the subcarrier configuration of the allocated resource units.

param nof_subcarrier
Number of subcarriers per resource unit The allowed values have dependencies, see 'Resource unit allocation'.

param offset
Offset of the first allocated subcarrier from the edge of the transmission bandwidth For a subcarrier spacing of 3.75 kHz / 15 kHz, n equals 48 / 12.

6.1.1.2 Prach

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:TOUT
CONFigure:NIOT:MEASurement<Instance>:PRACh:REPetition
CONFigure:NIOT:MEASurement<Instance>:PRACh:SCONdition
CONFigure:NIOT:MEASurement<Instance>:PRACh:MOEXception
CONFigure:NIOT:MEASurement<Instance>:PRACh:PFORmat
CONFigure:NIOT:MEASurement<Instance>:PRACh:NOPReambles
CONFigure:NIOT:MEASurement<Instance>:PRACh:POPReambles
```

class PrachCls

Prach commands group definition. 26 total commands, 4 Subgroups, 7 group commands

get_mo_exception() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MOEXception
value: bool = driver.configure.niotMeas.prach.get_mo_exception()
```

Specifies whether measurement results that the CMP180 identifies as faulty or inaccurate are rejected.

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

get_no_preambles() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:NOPReambles
value: int = driver.configure.niotMeas.prach.get_no_preambles()
```

Specifies the number of preambles to be captured per measurement interval.

return
number_preamble: No help available

get_pformat() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:PFORmat
value: int = driver.configure.niotMeas.prach.get_pformat()
```

Specifies the NPRACH preamble format used by the UE.

return
preamble_format: No help available

get_po_preambles() → PeriodPreamble

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:POPReambles
value: enums.PeriodPreamble = driver.configure.niotMeas.prach.get_po_preambles()
```

Specifies the periodicity of preambles to be captured for multi-preamble measurement squares.

return
period_preamble: Periodicity in ms

get_repetition() → Repeat

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:REPetition
value: enums.Repeat = driver.configure.niotMeas.prach.get_repetition()
```

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

return
repetition: SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

get_scondition() → StopCondition

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:SCONdition
value: enums.StopCondition = driver.configure.niotMeas.prach.get_scondition()
```

Qualifies whether the measurement is stopped after a failed limit check or continued. SLFail means that the measurement is stopped and reaches the RDY state when one of the results exceeds the limits.

return
stop_condition: NONE: Continue measurement irrespective of the limit check. SLFail: Stop measurement on limit failure.

get_timeout() → float

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:TOUT
value: float = driver.configure.niotMeas.prach.get_timeout()
```

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

return
timeout: No help available

set_mo_exception(*meas_on_exception: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MOEXception
driver.configure.niotMeas.prach.set_mo_exception(meas_on_exception = False)
```

Specifies whether measurement results that the CMP180 identifies as faulty or inaccurate are rejected.

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

set_no_preambles(*number_preamble: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:NOPReambles
driver.configure.niotMeas.prach.set_no_preambles(number_preamble = 1)
```

Specifies the number of preambles to be captured per measurement interval.

param number_preamble
No help available

set_pformat(*preamble_format: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:PFORmat
driver.configure.niotMeas.prach.set_pformat(preamble_format = 1)
```

Specifies the NPRACH preamble format used by the UE.

param preamble_format
No help available

set_po_preambles(*period_preamble: PeriodPreamble*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:POPReambles
driver.configure.niotMeas.prach.set_po_preambles(period_preamble = enums.
↪PeriodPreamble.MS160)
```

Specifies the periodicity of preambles to be captured for multi-preamble measurement squares.

param period_preamble
Periodicity in ms

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:REPetition
driver.configure.niotMeas.prach.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

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

param repetition
SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

set_scondition(*stop_condition: StopCondition*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:SCONdition
driver.configure.niotMeas.prach.set_scondition(stop_condition = enums.
↪StopCondition.NONE)
```

Qualifies whether the measurement is stopped after a failed limit check or continued. SLFail means that the measurement is stopped and reaches the RDY state when one of the results exceeds the limits.

param stop_condition
NONE: Continue measurement irrespective of the limit check. SLFail: Stop measurement on limit failure.

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:TOUT
driver.configure.niotMeas.prach.set_timeout(timeout = 1.0)
```

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

param timeout
No help available

Cloning the Group

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

Subgroups

6.1.1.2.1 Limit

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:FERRor
```

class LimitCls

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

get_freq_error() → float

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:FERRor
value: float or bool = driver.configure.niotMeas.prach.limit.get_freq_error()
```

Defines an upper limit for the carrier frequency error.

return
frequency_error: (float or boolean) No help available

set_freq_error(*frequency_error: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:FERRor
driver.configure.niotMeas.prach.limit.set_freq_error(frequency_error = 1.0)
```

Defines an upper limit for the carrier frequency error.

param frequency_error
(float or boolean) No help available

Cloning the Group

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

Subgroups

6.1.1.2.1.1 EvMagnitude

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:EVMagnitude
```

class EvMagnitudeCls

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

class EvMagnitudeStruct

Response structure. Fields:

- Rms: float or bool: No parameter help available
- Peak: float or bool: No parameter help available

get() → EvMagnitudeStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:EVMagnitude
value: EvMagnitudeStruct = driver.configure.niotMeas.prach.limit.evMagnitude.
↪get()
```

Defines upper limits for the RMS and peak values of the error vector magnitude (EVM) .

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

set(*rms: float*, *peak: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:EVMagnitude
driver.configure.niotMeas.prach.limit.evMagnitude.set(rms = 1.0, peak = 1.0)
```

Defines upper limits for the RMS and peak values of the error vector magnitude (EVM) .

param rms
(float or boolean) No help available

param peak
(float or boolean) No help available

6.1.1.2.1.2 Merror

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:MERRor
```

class MerrorCls

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

class MerrorStruct

Response structure. Fields:

- Rms: float or bool: No parameter help available
- Peak: float or bool: No parameter help available

get() → MerrorStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:MERRor
value: MerrorStruct = driver.configure.niotMeas.prach.limit.merror.get()
```

Defines upper limits for the RMS and peak values of the magnitude error.

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

set(*rms: float*, *peak: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:MERRor
driver.configure.niotMeas.prach.limit.merror.set(rms = 1.0, peak = 1.0)
```

Defines upper limits for the RMS and peak values of the magnitude error.

param rms
(float or boolean) No help available

param peak
(float or boolean) No help available

6.1.1.2.1.3 Pdynamics

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:PDYNamics
```

class PdynamicsCls

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

class PdynamicsStruct

Response structure. Fields:

- Enable: bool: OFF: disables the limit check ON: enables the limit check
- On_Power_Upper: float: Upper limit for the ON power
- On_Power_Lower: float: Lower limit for the ON power
- Off_Power_Upper: float: Upper limit for the OFF power

get() → PdynamicsStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:PDYNamics
value: PdynamicsStruct = driver.configure.niotMeas.prach.limit.pdynamics.get()
```

Defines limits for the ON power and OFF power determined with the power dynamics measurement.

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

set(*enable: bool*, *on_power_upper: float*, *on_power_lower: float*, *off_power_upper: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:PDYNamics
driver.configure.niotMeas.prach.limit.pdynamics.set(enable = False, on_power_
↪upper = 1.0, on_power_lower = 1.0, off_power_upper = 1.0)
```

Defines limits for the ON power and OFF power determined with the power dynamics measurement.

param enable
OFF: disables the limit check ON: enables the limit check

param on_power_upper
Upper limit for the ON power

param on_power_lower
Lower limit for the ON power

param off_power_upper
Upper limit for the OFF power

6.1.1.2.1.4 Perror

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:PERRor
```

class PerrorCls

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

class PerrorStruct

Response structure. Fields:

- Rms: float or bool: No parameter help available
- Peak: float or bool: No parameter help available

get() → PerrorStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:PERRor
value: PerrorStruct = driver.configure.niotMeas.prach.limit.perror.get()
```

Defines symmetric limits for the RMS and peak values of the phase error. The limit check fails if the absolute value of the measured phase error exceeds the specified values.

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

set(*rms: float*, *peak: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:LIMit:PERRor
driver.configure.niotMeas.prach.limit.perror.set(rms = 1.0, peak = 1.0)
```

Defines symmetric limits for the RMS and peak values of the phase error. The limit check fails if the absolute value of the measured phase error exceeds the specified values.

param rms
(float or boolean) No help available

param peak
(float or boolean) No help available

6.1.1.2.2 Modulation

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWPosition
```

class ModulationCls

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

get_ew_position() → LowHigh

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWPosition
value: enums.LowHigh = driver.configure.niotMeas.prach.modulation.get_ew_
↪position()
```

Specifies the position of the EVM window used for calculation of the trace results.

return
evm_window_pos: No help available

set_ew_position(*evm_window_pos: LowHigh*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWPosition
driver.configure.niotMeas.prach.modulation.set_ew_position(evm_window_pos =␣
↪enums.LowHigh.HIGH)
```

Specifies the position of the EVM window used for calculation of the trace results.

param evm_window_pos
No help available

Cloning the Group

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

Subgroups

6.1.1.2.2.1 EwLength

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWLength
```

class EwLengthCls

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

get_value() → List[int]

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWLength
value: List[int] = driver.configure.niotMeas.prach.modulation.ewLength.get_
↪value()
```

Specifies the EVM window length in samples for all preamble formats.

return
evm_window_length: No help available

set_value(*evm_window_length: List[int]*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWLength
driver.configure.niotMeas.prach.modulation.ewLength.set_value(evm_window_length␣
↪= [1, 2, 3])
```

Specifies the EVM window length in samples for all preamble formats.

param evm_window_length
No help available

Cloning the Group

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

Subgroups

6.1.1.2.2.2 Pformat<PreambleFormat>

RepCap Settings

```
# Range: Fmt1 .. Fmt2
rc = driver.configure.niotMeas.prach.modulation.ewLength.pformat.repcap_preambleFormat_
↪get()
driver.configure.niotMeas.prach.modulation.ewLength.pformat.repcap_preambleFormat_
↪set(repcap.PreambleFormat.Fmt1)
```

SCPI Command :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWLength:PFORmat<PreambleFormat>
```

class PformatCls

Pformat commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: PreambleFormat, default value after init: PreambleFormat.Fmt1

get(*preambleFormat=PreambleFormat.Default*) → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWLength:PFORmat
↪<PreambleFormat>
value: int = driver.configure.niotMeas.prach.modulation.ewLength.pformat.
↪get(preambleFormat = repcap.PreambleFormat.Default)
```

No command help available

param preambleFormat
optional repeated capability selector. Default value: Fmt1 (settable in the interface 'Pformat')

return
evm_window_length: No help available

set(*evm_window_length: int*, *preambleFormat=PreambleFormat.Default*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:MODulation:EWLength:PFORmat
↪<PreambleFormat>
driver.configure.niotMeas.prach.modulation.ewLength.pformat.set(evm_window_
↪length = 1, preambleFormat = repcap.PreambleFormat.Default)
```

No command help available

param evm_window_length
No help available

param preambleFormat
optional repeated capability selector. Default value: Fmt1 (settable in the interface 'Pformat')

Cloning the Group

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

6.1.1.2.3 Result

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult[:ALL]
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:EVMagnitude
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:EVPReamble
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:MERRor
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PERRor
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:IQ
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PDYNamics
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PVPReamble
CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:TXM
```

class ResultCls

Result commands group definition. 9 total commands, 0 Subgroups, 9 group commands

class AllStruct

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

- Evm: bool: Error vector magnitude. OFF: Do not evaluate results. ON: Evaluate results
- Magnitude_Error: bool: Magnitude error
- Phase_Error: bool: Phase error
- Iq: bool: I/Q constellation diagram
- Power_Dynamics: bool: Power dynamics
- Tx_Measurement: bool: Statistical overview
- Evm_Vs_Preamble: bool: Optional setting parameter. Error vector magnitude vs preamble
- Power_Vs_Preamble: bool: Optional setting parameter. Power vs preamble

get_all() → AllStruct

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult[:ALL]
value: AllStruct = driver.configure.niotMeas.prach.result.get_all()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. This command combines all other CONFigure:NIOT:MEAS<i>:PRACh:RESult... commands.

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

get_ev_magnitude() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:EVMagnitude
value: bool = driver.configure.niotMeas.prach.result.get_ev_magnitude()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_ev_preamble() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:EVPReamble
value: bool = driver.configure.niotMeas.prach.result.get_ev_preamble()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_iq() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:IQ
value: bool = driver.configure.niotMeas.prach.result.get_iq()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_merror() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:MERRor
value: bool = driver.configure.niotMeas.prach.result.get_merror()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_pdynamics() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PDYNamics
value: bool = driver.configure.niotMeas.prach.result.get_pdynamics()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_perror() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PERRor
value: bool = driver.configure.niotMeas.prach.result.get_perror()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_pv_preamble() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PVPReamble
value: bool = driver.configure.niotMeas.prach.result.get_pv_preamble()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

get_txm() → bool

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:TXM
value: bool = driver.configure.niotMeas.prach.result.get_txm()
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

return
enable: OFF: Do not evaluate results. ON: Evaluate results.

set_all(*value: AllStruct*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult[:ALL]
structure = driver.configure.niotMeas.prach.result.AllStruct()
structure.Evm: bool = False
structure.Magnitude_Error: bool = False
structure.Phase_Error: bool = False
structure.Iq: bool = False
structure.Power_Dynamics: bool = False
structure.Tx_Measurement: bool = False
structure.Evm_Vs_Preamble: bool = False
structure.Power_Vs_Preamble: bool = False
driver.configure.niotMeas.prach.result.set_all(value = structure)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. This command combines all other CONFigure:NIOT:MEAS<i>:PRACh:RESult... commands.

param value
see the help for AllStruct structure arguments.

set_ev_magnitude(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:EVMagnitude
driver.configure.niotMeas.prach.result.set_ev_magnitude(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_ev_preamble(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:EVPReamble
driver.configure.niotMeas.prach.result.set_ev_preamble(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_iq(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:IQ
driver.configure.niotMeas.prach.result.set_iq(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_merror(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:MERRor
driver.configure.niotMeas.prach.result.set_merror(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_pdynamics(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PDYNamics
driver.configure.niotMeas.prach.result.set_pdynamics(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_perror(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PERRor
driver.configure.niotMeas.prach.result.set_perror(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_pv_preamble(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:PVPReamble
driver.configure.niotMeas.prach.result.set_pv_preamble(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

set_txm(*enable: bool*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:RESult:TXM
driver.configure.niotMeas.prach.result.set_txm(enable = False)
```

Enables or disables the evaluation of results of NB-IoT PRACH measurements. The mnemonic after 'RESult' denotes the measurement type: error vector magnitude, magnitude error, phase error, I/Q constellation diagram, power dynamics, TX measurement statistical overview, error vector magnitude vs preamble, power vs preamble For reset values, see method RsCMPX_NiotMeas.Configure.NiotMeas.Prach.Result.all.

param enable
OFF: Do not evaluate results. ON: Evaluate results.

6.1.1.2.4 Scount

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:PRACh:SCOunt:MODulation
CONFigure:NIOT:MEASurement<Instance>:PRACh:SCOunt:PDYNamics
```

class ScountCls

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

get_modulation() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:SCOunt:MODulation
value: int = driver.configure.niotMeas.prach.scount.get_modulation()
```

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

return
statistic_count: Number of measurement intervals

get_pdynamics() → int

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:SCOunt:PDYNamics
value: int = driver.configure.niotMeas.prach.scount.get_pdynamics()
```

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

return
statistic_count: Number of measurement intervals

set_modulation(*statistic_count: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:SCOunt:MODulation
driver.configure.niotMeas.prach.scount.set_modulation(statistic_count = 1)
```

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

param statistic_count
Number of measurement intervals

set_pdynamics(*statistic_count: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:PRACh:SCOunt:PDYNamics
driver.configure.niotMeas.prach.scount.set_pdynamics(statistic_count = 1)
```

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

param statistic_count
Number of measurement intervals

6.1.1.3 RfSettings

SCPI Commands :

```
CONFigure:NIOT:MEASurement<Instance>:RFSettings:EATTenuation
CONFigure:NIOT:MEASurement<Instance>:RFSettings:UMARgin
CONFigure:NIOT:MEASurement<Instance>:RFSettings:ENPower
CONFigure:NIOT:MEASurement<Instance>:RFSettings:FREQuency
CONFigure:NIOT:MEASurement<Instance>:RFSettings:FOFFset
CONFigure:NIOT:MEASurement<Instance>:RFSettings:MLOFfset
```

class RfSettingsCls

RfSettings commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_eattenuation() → float

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

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

return
rf_input_ext_att: No help available

get_envelope_power() → float

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

Sets the expected nominal power of the measured RF signal.

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

get_foffset() → int

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

No command help available

return
offset: No help available

get_frequency() → float

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

Selects the center frequency of the RF analyzer. Using the unit CH, the frequency can be set via the channel number. The allowed channel number range depends on the operating band, see 'Frequency bands'. For the supported frequency range, see 'Frequency ranges'.

return
analyzer_freq: No help available

get_ml_offset() → float

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

No command help available

return
mix_lev_offset: No help available

get_umargin() → float

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:RFSettings:UMARgin
value: float = driver.configure.niotMeas.rfSettings.get_umargin()
```

Sets the margin that the measurement adds to the expected nominal power to determine the reference power. The reference power minus the external input attenuation must be within the power range of the selected input connector. Refer to the specifications document.

return
user_margin: No help available

set_eattenuation(*rf_input_ext_att: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:RFSettings:EATTenuation
driver.configure.niotMeas.rfSettings.set_eattenuation(rf_input_ext_att = 1.0)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

param rf_input_ext_att
No help available

set_envelope_power(*exp_nom_pow: float*) → None

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

Sets the expected nominal power of the measured RF signal.

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

set_foffset(*offset: int*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:RFSettings:FOFFset
driver.configure.niotMeas.rfSettings.set_foffset(offset = 1)
```

No command help available

param offset
No help available

set_frequency(*analyzer_freq: float*) → None

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

Selects the center frequency of the RF analyzer. Using the unit CH, the frequency can be set via the channel number. The allowed channel number range depends on the operating band, see 'Frequency bands'. For the supported frequency range, see 'Frequency ranges'.

param analyzer_freq
No help available

set_ml_offset(*mix_lev_offset: float*) → None

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

No command help available

param mix_lev_offset
No help available

set_umargin(*user_margin: float*) → None

```
# SCPI: CONFigure:NIOT:MEASurement<Instance>:RFSettings:UMARgin
driver.configure.niotMeas.rfSettings.set_umargin(user_margin = 1.0)
```

Sets the margin that the measurement adds to the expected nominal power to determine the reference power. The reference power minus the external input attenuation must be within the power range of the selected input connector. Refer to the specifications document.

param user_margin
No help available

6.2 NiotMeas

class NiotMeasCls

NiotMeas commands group definition. 204 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.1 MultiEval

SCPI Commands :

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

class MultiEvalCls

MultiEval commands group definition. 141 total commands, 11 Subgroups, 3 group commands

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

```
# SCPI: ABORt:NIOT:MEASurement<Instance>:MEValuation
driver.niotMeas.multiEval.abort()
```

(continues on next page)

(continued from previous page)

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

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: INITiate:NIOT:MEASurement<Instance>:MEValuation
driver.niotMeas.multiEval.initiate()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

stop() → None

```
# SCPI: STOP:NIOT:MEASurement<Instance>:MEValuation
driver.niotMeas.multiEval.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

```
# SCPI: STOP:NIOT:MEASurement<Instance>:MEValuation
driver.niotMeas.multiEval.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as stop, but waits for the operation to complete before continuing further. Use the RsCMPX_NiotMeas.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.niotMeas.multiEval.clone()
```

Subgroups

6.2.1.1 Aclr

class AclrCls

Aclr 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.niotMeas.multiEval.aclr.clone()
```

Subgroups

6.2.1.1.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:ACLR:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:ACLR:AVERage
CALCulate:NIOT:MEASurement<Instance>:MEValuation:ACLR:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Utra_Neg: enums.ResultStatus2: ACLR for the adjacent UTRA channel with lower frequency
- Gsm_Neg: enums.ResultStatus2: ACLR for the adjacent GSM channel with lower frequency
- Nb_Iot: enums.ResultStatus2: Power in the allocated NB-IOT channel
- Gsm_Pos: enums.ResultStatus2: ACLR for the adjacent GSM channel with higher frequency
- Utra_Pos: enums.ResultStatus2: ACLR for the adjacent UTRA channel with higher frequency

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Utra_Neg: float: ACLR for the adjacent UTRA channel with lower frequency
- Gsm_Neg: float: ACLR for the adjacent GSM channel with lower frequency
- Nb_Iot: float: Power in the allocated NB-IOT channel
- Gsm_Pos: float: ACLR for the adjacent GSM channel with higher frequency
- Utra_Pos: float: ACLR for the adjacent UTRA channel with higher frequency

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:ACLR:AVERage
value: CalculateStruct = driver.niotMeas.multiEval.aclr.average.calculate()
```

Returns the relative ACLR values as displayed in the table below the ACLR diagram. The current and average values can be retrieved.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:ACLR:AVERage
value: ResultData = driver.niotMeas.multiEval.aclr.average.fetch()
```

Returns the relative ACLR values as displayed in the table below the ACLR diagram. The current and average values can be retrieved.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:ACLR:AVERage
value: ResultData = driver.niotMeas.multiEval.aclr.average.read()
```

Returns the relative ACLR values as displayed in the table below the ACLR diagram. The current and average values can be retrieved.

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

6.2.1.1.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:ACLR:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:ACLR:CURRent
CALCulate:NIOT:MEASurement<Instance>:MEValuation:ACLR:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Utra_Neg: float: ACLR for the adjacent UTRA channel with lower frequency
- Gsm_Neg: float: ACLR for the adjacent GSM channel with lower frequency
- Nb_Iot: float: Power in the allocated NB-IOT channel
- Gsm_Pos: float: ACLR for the adjacent GSM channel with higher frequency
- Utra_Pos: float: ACLR for the adjacent UTRA channel with higher frequency

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Utra_Neg: float: ACLR for the adjacent UTRA channel with lower frequency
- Gsm_Neg: float: ACLR for the adjacent GSM channel with lower frequency
- Nb_Iot: float: Power in the allocated NB-IOT channel
- Gsm_Pos: float: ACLR for the adjacent GSM channel with higher frequency
- Utra_Pos: float: ACLR for the adjacent UTRA channel with higher frequency

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:ACLR:CURRent
value: CalculateStruct = driver.niotMeas.multiEval.aclr.current.calculate()
```

Returns the relative ACLR values as displayed in the table below the ACLR diagram. The current and average values can be retrieved.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:ACLR:CURRent
value: ResultData = driver.niotMeas.multiEval.aclr.current.fetch()
```

Returns the relative ACLR values as displayed in the table below the ACLR diagram. The current and average values can be retrieved.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:ACLR:CURRent
value: ResultData = driver.niotMeas.multiEval.aclr.current.read()
```

Returns the relative ACLR values as displayed in the table below the ACLR diagram. The current and average values can be retrieved.

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

6.2.1.2 EvMagnitude

class EvMagnitudeCls

EvMagnitude 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.niotMeas.multiEval.evMagnitude.clone()
```

Subgroups

6.2.1.2.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:AVERage
value: List[float] = driver.niotMeas.multiEval.evMagnitude.average.fetch()
```

Returns the values of the EVM RMS bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:AVERage
value: List[float] = driver.niotMeas.multiEval.evMagnitude.average.read()
```

Returns the values of the EVM RMS bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.2.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:CURRent
value: List[float] = driver.niotMeas.multiEval.evMagnitude.current.fetch()
```

Returns the values of the EVM RMS bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:CURRent
value: List[float] = driver.niotMeas.multiEval.evMagnitude.current.read()
```

Returns the values of the EVM RMS bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.2.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:MAXimum
value: List[float] = driver.niotMeas.multiEval.evMagnitude.maximum.fetch()
```

Returns the values of the EVM RMS bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:MAXimum
value: List[float] = driver.niotMeas.multiEval.evMagnitude.maximum.read()
```

Returns the values of the EVM RMS bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.2.4 Peak

class PeakCls

Peak 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.niotMeas.multiEval.evMagnitude.peak.clone()
```

Subgroups

6.2.1.2.4.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:AVERage
value: List[float] = driver.niotMeas.multiEval.evMagnitude.peak.average.fetch()
```

Returns the values of the EVM peak bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:AVERage
value: List[float] = driver.niotMeas.multiEval.evMagnitude.peak.average.read()
```

Returns the values of the EVM peak bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.2.4.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:CURRent
value: List[float] = driver.niotMeas.multiEval.evMagnitude.peak.current.fetch()
```

Returns the values of the EVM peak bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:CURRent
value: List[float] = driver.niotMeas.multiEval.evMagnitude.peak.current.read()
```

Returns the values of the EVM peak bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.2.4.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:MAXimum
value: List[float] = driver.niotMeas.multiEval.evMagnitude.peak.maximum.fetch()
```

Returns the values of the EVM peak bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:EVMagnitude:PEAK:MAXimum
value: List[float] = driver.niotMeas.multiEval.evMagnitude.peak.maximum.read()
```

Returns the values of the EVM peak bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Square EVM'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.3 InbandEmission

class InbandEmissionCls

InbandEmission 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.niotMeas.multiEval.inbandEmission.clone()
```

Subgroups

6.2.1.3.1 Margin

class MarginCls

Margin commands group definition. 6 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.1.3.1.1 Average

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:AVERage
```

class AverageCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified inband emission limits.
- Margin: float: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:AVERage
value: FetchStruct = driver.niotMeas.multiEval.inbandEmission.margin.average.
↪fetch()
```

Return the limit line margin results for the inband emissions. The CURRent margin indicates the minimum (vertical) distance between the inband emissions limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins. The margin results cannot be displayed at the GUI.

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

6.2.1.3.1.2 Current

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:CURRent
```

class CurrentCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified inband emission limits.
- Margin: float: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:CURRent
value: FetchStruct = driver.niotMeas.multiEval.inbandEmission.margin.current.
↪fetch()
```

Return the limit line margin results for the inband emissions. The CURRent margin indicates the minimum (vertical) distance between the inband emissions limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins. The margin results cannot be displayed at the GUI.

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.niotMeas.multiEval.inbandEmission.margin.current.clone()
```

Subgroups

6.2.1.3.1.3 ScIndex

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:CURRent:SCINdex
```

class ScIndexCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified inband emission limits.
- Sc_Index: int: Subcarrier index

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:IEMission:MARGin:CURRent:SCINdex
value: FetchStruct = driver.niotMeas.multiEval.inbandEmission.margin.current.
↪scIndex.fetch()
```

Return subcarrier indices for inband emission margins. At these SC indices, the CURRent and EXTReme margins have been detected (see method RsCMPX_NiotMeas.NiotMeas.MultiEval.InbandEmission.Margin.Current.fetch and ...:EXTReme) .

return

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

6.2.1.3.1.4 Extreme

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:EXTReme
```

class ExtremeCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified inband emission limits.
- Margin: float: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:EXTReme
value: FetchStruct = driver.niotMeas.multiEval.inbandEmission.margin.extreme.
↪fetch()
```

Return the limit line margin results for the inband emissions. The CURRent margin indicates the minimum (vertical) distance between the inband emissions limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins. The margin results cannot be displayed at the GUI.

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.niotMeas.multiEval.inbandEmission.margin.extreme.clone()
```

Subgroups

6.2.1.3.1.5 ScIndex

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:EXTReme:SCINdex
```

class ScIndexCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified inband emission limits.
- Sc_Index: int: Subcarrier index

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:IEMission:MARGin:EXTReme:SCINdex
value: FetchStruct = driver.niotMeas.multiEval.inbandEmission.margin.extreme.
↪scIndex.fetch()
```

Return subcarrier indices for inband emission margins. At these SC indices, the CURRent and EXTReme margins have been detected (see method RsCMPX_NiotMeas.NiotMeas.MultiEval.InbandEmission.Margin.Current.fetch and ...:EXTReme) .

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

6.2.1.3.1.6 StandardDev

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:SDEViation
```

class StandardDevCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified inband emission limits.
- Margin: float: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:IEMission:MARGin:SDEViation
value: FetchStruct = driver.niotMeas.multiEval.inbandEmission.margin.
↪standardDev.fetch()
```

Return the limit line margin results for the inband emissions. The CURRent margin indicates the minimum (vertical) distance between the inband emissions limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins. The margin results cannot be displayed at the GUI.

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

6.2.1.4 ListPy

class ListPyCls

ListPy commands group definition. 24 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.1.4.1 Segment

RepCap Settings

```
# Range: Nr1 .. Nr2000
rc = driver.niotMeas.multiEval.listPy.segment.repcap_segment_get()
driver.niotMeas.multiEval.listPy.segment.repcap_segment_set(repcap.Segment.Nr1)
```

class SegmentCls

Segment commands group definition. 23 total commands, 4 Subgroups, 0 group commands Repeated Capability: Segment, default value after init: Segment.Nr1

Cloning the Group

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

Subgroups

6.2.1.4.1.1 Aclr

class AclrCls

Aclr 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.niotMeas.multiEval.listPy.segment.aclr.clone()
```

Subgroups

6.2.1.4.1.2 Average

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:ACLR:AVERage
```

class AverageCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Utra_Neg: float: ACLR for the adjacent UTRA channel with lower frequency
- Gsm_Neg: float: ACLR for the adjacent GSM channel with lower frequency
- Nb_Iot: float: Power in the NB-IoT channel
- Gsm_Pos: float: ACLR for the adjacent GSM channel with higher frequency
- Utra_Pos: float: ACLR for the adjacent UTRA channel with higher frequency

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:ACLR:AVERage
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.aclr.average.
↪fetch(segment = repcap.Segment.Default)
```

Return ACLR single value results for segment <no> in list mode.

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

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

6.2.1.4.1.3 Current

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:ACLR:CURRent
```

class CurrentCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Utra_Neg: float: ACLR for the adjacent UTRA channel with lower frequency
- Gsm_Neg: float: ACLR for the adjacent GSM channel with lower frequency
- Nb_Iot: float: Power in the NB-IoT channel
- Gsm_Pos: float: ACLR for the adjacent GSM channel with higher frequency
- Utra_Pos: float: ACLR for the adjacent UTRA channel with higher frequency

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:ACLR:CURRent
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.aclr.current.
↪fetch(segment = repcap.Segment.Default)
```

Return ACLR single value results for segment <no> in list mode.

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

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

6.2.1.4.1.4 InbandEmission

class InbandEmissionCls

InbandEmission 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.niotMeas.multiEval.listPy.segment.inbandEmission.clone()
```

Subgroups

6.2.1.4.1.5 Margin

class MarginCls

Margin commands group definition. 6 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.niotMeas.multiEval.listPy.segment.inbandEmission.margin.clone()
```

Subgroups

6.2.1.4.1.6 Average

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:IEMission:MARGin:AVERage
```

class AverageCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin: float: No parameter help available

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:AVERage
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.inbandEmission.
↪margin.average.fetch(segment = repcap.Segment.Default)
```

Return the inband emission limit line margin results for segment <no> in list mode. The CURRent margins indicate the minimum (vertical) distance between the limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins.

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

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

6.2.1.4.1.7 Current

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:IEMission:MARGin:CURRent
```

class CurrentCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin: float: No parameter help available

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:CURRent
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.inbandEmission.
↪margin.current.fetch(segment = repcap.Segment.Default)
```

Return the inband emission limit line margin results for segment <no> in list mode. The CURRent margins indicate the minimum (vertical) distance between the limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins.

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

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.niotMeas.multiEval.listPy.segment.inbandEmission.margin.current.clone()
```

Subgroups

6.2.1.4.1.8 ScIndex

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:CURRent:SCINdex
```

class ScIndexCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Sc_Index: int: Subcarrier index

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:CURRent:SCINdex
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.inbandEmission.
↪margin.current.scIndex.fetch(segment = repcap.Segment.Default)
```

Return subcarrier indices for inband emission margins for segment <no> in list mode. At these SC indices, the CURRent and EXTReme margins have been detected.

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

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

6.2.1.4.1.9 Extreme

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:IEMission:MARGin:EXTReme
```

class ExtremeCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment

- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin: float: No parameter help available

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:EXTReme
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.inbandEmission.
↪margin.extreme.fetch(segment = repcap.Segment.Default)
```

Return the inband emission limit line margin results for segment <no> in list mode. The CURRent margins indicate the minimum (vertical) distance between the limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins.

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

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.niotMeas.multiEval.listPy.segment.inbandEmission.margin.extreme.clone()
```

Subgroups

6.2.1.4.1.10 ScIndex

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:EXTReme:SCINdex
```

class ScIndexCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Sc_Index: int: Subcarrier index

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:EXTReme:SCINdex
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.inbandEmission.
↪margin.extreme.scIndex.fetch(segment = repcap.Segment.Default)
```

Return subcarrier indices for inband emission margins for segment <no> in list mode. At these SC indices, the CURRent and EXTReme margins have been detected.

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

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

6.2.1.4.1.11 StandardDev

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:IEMission:MARGin:SDEViation
```

class StandardDevCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin: float: No parameter help available

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:IEMission:MARGin:SDEViation
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.inbandEmission.
↪margin.standardDev.fetch(segment = repcap.Segment.Default)
```

Return the inband emission limit line margin results for segment <no> in list mode. The CURRent margins indicate the minimum (vertical) distance between the limit line and the current trace. A negative result indicates that the limit is exceeded. The AVERage, EXTReme and SDEViation values are calculated from the current margins.

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

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

6.2.1.4.1.12 Modulation

class ModulationCls

Modulation 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.niotMeas.multiEval.listPy.segment.modulation.clone()
```

Subgroups

6.2.1.4.1.13 Average

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:MODulation:AVERage
```

class AverageCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Evm_Rms: float: Error vector magnitude RMS value
- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value
- Iq_Offset: float: I/Q origin offset
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power
- Sc_Power: float: Power in allocated subcarriers
- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value

- Ph_Error_Dmrs: float: Phase error DMRS value

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:MODulation:AVERage
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.modulation.
↪average.fetch(segment = repcap.Segment.Default)
```

Returns current, average and standard deviation modulation single value results for segment <no> in list mode.

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

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

6.2.1.4.1.14 Current

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:MODulation:CURRent
```

class CurrentCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Evm_Rms: float: Error vector magnitude RMS value
- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value
- Iq_Offset: float: I/Q origin offset
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power
- Sc_Power: float: Power in allocated subcarriers

- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value
- Ph_Error_Dmrs: float: Phase error DMRS value

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:MODulation:CURRent
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.modulation.
↪current.fetch(segment = repcap.Segment.Default)
```

Returns current, average and standard deviation modulation single value results for segment <no> in list mode.

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

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

6.2.1.4.1.15 Extreme

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:MODulation:EXTReme
```

class ExtremeCls

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

class FetchStruct

Response structure. Fields:

- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Evm_Rms: float: Error vector magnitude RMS value
- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value
- Iq_Offset: float: I/Q origin offset
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power_Minimum: float: Minimum user equipment power
- Tx_Power_Maximum: float: Maximum user equipment power

- Peak_Power_Min: float: Minimum user equipment peak power
- Peak_Power_Max: float: Maximum user equipment peak power
- Sc_Pow_Min: float: No parameter help available
- Sc_Pow_Max: float: No parameter help available
- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value
- Ph_Error_Dmrs: float: Phase error DMRS value

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:MODulation:EXTReme
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.modulation.
↪extreme.fetch(segment = repcap.Segment.Default)
```

Returns extreme modulation single value results for segment <no> in list mode.

Suppressed linked return values: reliability

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

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

6.2.1.4.1.16 StandardDev

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:MODulation:SDEViation
```

class StandardDevCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Evm_Rms: float: Error vector magnitude RMS value
- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value

- Iq_Offset: float: I/Q origin offset
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power
- Sc_Power: float: Power in allocated subcarriers
- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value
- Ph_Error_Dmrs: float: Phase error DMRS value

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:MODulation:SDEViation
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.modulation.
↪standardDev.fetch(segment = repcap.Segment.Default)
```

Returns current, average and standard deviation modulation single value results for segment <no> in list mode.

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

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

6.2.1.4.1.17 SeMask

class SeMaskCls

SeMask commands group definition. 11 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.niotMeas.multiEval.listPy.segment.seMask.clone()
```

Subgroups

6.2.1.4.1.18 Average

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask:AVERage
```

class AverageCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Obw: float: Occupied bandwidth
- Tx_Power: float: Total TX power within 400 kHz

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:AVERage
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.average.
↪fetch(segment = repcap.Segment.Default)
```

Return spectrum emission single value results for segment <no> in list mode.

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

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

6.2.1.4.1.19 Current

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask:CURRent
```

class CurrentCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Obw: float: Occupied bandwidth
- Tx_Power: float: Total TX power within 400 kHz

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:CURRent
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.current.
↪fetch(segment = repcap.Segment.Default)
```

Return spectrum emission single value results for segment <no> in list mode.

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

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

6.2.1.4.1.20 Extreme

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask:EXTReme
```

class ExtremeCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Obw: float: Occupied bandwidth
- Tx_Power_Min: float: Minimum total TX power
- Tx_Power_Max: float: Maximum total TX power

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:EXTReme
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.extreme.
↪fetch(segment = repcap.Segment.Default)
```

Returns spectrum emission extreme results for segment <no> in list mode.

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

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

6.2.1.4.1.21 Margin

class MarginCls

Margin commands group definition. 7 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.niotMeas.multiEval.listPy.segment.seMask.margin.clone()
```

Subgroups

6.2.1.4.1.22 All

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask:MARGin:ALL
```

class AllCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin_Curr_Neg: List[float]: Comma-separated list of 12 margin results For the current trace, area 1 to 12, negative frequency offsets
- Margin_Curr_Pos: List[float]: Comma-separated list of 12 margin results For the current trace, area 1 to 12, positive frequency offsets
- Margin_Avg_Neg: List[float]: Comma-separated list of 12 margin results For the average trace, area 1 to 12, negative frequency offsets
- Margin_Avg_Pos: List[float]: Comma-separated list of 12 margin results For the average trace, area 1 to 12, positive frequency offsets
- Margin_Min_Neg: List[float]: Comma-separated list of 12 margin results For the maximum trace, area 1 to 12, negative frequency offsets
- Margin_Min_Pos: List[float]: Comma-separated list of 12 margin results For the maximum trace, area 1 to 12, positive frequency offsets

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:ALL
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.margin.all.
↪fetch(segment = repcap.Segment.Default)
```

Returns limit line margin values, i.e. vertical distances between the spectrum emission mask limit line and a trace, for segment <no> in list mode.

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

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

6.2.1.4.1.23 Average

class AverageCls

Average 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.niotMeas.multiEval.listPy.segment.seMask.margin.average.clone()
```

Subgroups

6.2.1.4.1.24 Negativ

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:AVERage:NEGativ
```

class NegativCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin_Avg_Neg_X: List[float]: No parameter help available
- Margin_Avg_Neg_Y: List[float]: No parameter help available

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:AVERage:NEGativ
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.margin.
↪average.negativ.fetch(segment = repcap.Segment.Default)
```

Return spectrum emission mask margin results for segment <no> in list mode. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <SegReliability>, <StatistExpired>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {... }area 12 For inactive areas, NCAP is returned.

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

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

6.2.1.4.1.25 Positiv

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:AVERage:POSitiv
```

class PositivCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin_Avg_Pos_X: List[float]: X-position of margin
- Margin_Avg_Pos_Y: List[float]: Y-value of margin

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:AVERage:POSitiv
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.margin.
↪average.positiv.fetch(segment = repcap.Segment.Default)
```

Return spectrum emission mask margin results for segment <no> in list mode. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <SegReliability>, <StatistExpired>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {... }area 12 For inactive areas, NCAP is returned.

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

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

6.2.1.4.1.26 Current

class CurrentCls

Current 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.niotMeas.multiEval.listPy.segment.seMask.margin.current.clone()
```

Subgroups

6.2.1.4.1.27 Negativ

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:CURRent:NEGativ
```

class NegativCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin_Curr_Neg_X: List[float]: No parameter help available
- Margin_Curr_Neg_Y: List[float]: No parameter help available

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:CURRent:NEGativ
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.margin.
↪current.negativ.fetch(segment = repcap.Segment.Default)
```

Return spectrum emission mask margin results for segment <no> in list mode. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <SegReliability>, <StatistExpired>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {... }area 12 For inactive areas, NCAP is returned.

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

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

6.2.1.4.1.28 Positiv

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:CURRent:POSitiv
```

class PositivCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin_Curr_Pos_X: List[float]: X-position of margin
- Margin_Curr_Pos_Y: List[float]: Y-value of margin

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:CURRent:POSitiv
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.margin.
↪current.positiv.fetch(segment = repcap.Segment.Default)
```

Return spectrum emission mask margin results for segment <no> in list mode. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <SegReliability>, <StatistExpired>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {... }area 12 For inactive areas, NCAP is returned.

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

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

6.2.1.4.1.29 Minimum

class MinimumCls

Minimum 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.niotMeas.multiEval.listPy.segment.seMask.margin.minimum.clone()
```

Subgroups

6.2.1.4.1.30 Negativ

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:MINimum:NEGativ
```

class NegativCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin_Min_Neg_X: List[float]: No parameter help available
- Margin_Min_Neg_Y: List[float]: No parameter help available

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:MINimum:NEGativ
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.margin.
↪minimum.negativ.fetch(segment = repcap.Segment.Default)
```

Return spectrum emission mask margin results for segment <no> in list mode. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <SegReliability>, <StatistExpired>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {... }area 12 For inactive areas, NCAP is returned.

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

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

6.2.1.4.1.31 Positiv

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:MINimum:POSitiv
```

class PositivCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Margin_Min_Pos_X: List[float]: X-position of margin
- Margin_Min_Pos_Y: List[float]: Y-value of margin

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:MARGin:MINimum:POSitiv
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.margin.
↪minimum.positiv.fetch(segment = repcap.Segment.Default)
```

Return spectrum emission mask margin results for segment <no> in list mode. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <SegReliability>, <StatistExpired>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {... }area 12 For inactive areas, NCAP is returned.

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

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

6.2.1.4.1.32 StandardDev

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>:SEMask:SDEViation
```

class StandardDevCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Seg_Reliability: int: Reliability indicator for the segment
- Statist_Expired: int: Reached statistical length in slots
- Out_Of_Tolerance: int: Percentage of measured slots with failed limit check
- Obw: float: Occupied bandwidth
- Tx_Power: float: Total TX power within 400 kHz

fetch(*segment=Segment.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SEGMent<nr>
↪:SEMask:SDEViation
value: FetchStruct = driver.niotMeas.multiEval.listPy.segment.seMask.
↪standardDev.fetch(segment = repcap.Segment.Default)
```

Return spectrum emission single value results for segment <no> in list mode.

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

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

6.2.1.4.2 Sreliability

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SRELiability
```

class SreliabilityCls

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

fetch() → List[int]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:LIST:SRELiability
value: List[int] = driver.niotMeas.multiEval.listPy.sreliability.fetch()
```

Returns the segment reliability for all measured list mode segments. A common reliability indicator of zero indicates that the results in all measured segments are valid. A non-zero value indicates that an error occurred in at least one of the measured segments. If you get a non-zero common reliability indicator,

you can use this command to retrieve the individual reliability values of all measured segments for further analysis.

Suppressed linked return values: reliability

return
seg_reliability: Comma-separated list of values, one per measured segment The meaning of the returned values is the same as for the common reliability indicator, see previous parameter.

6.2.1.5 Merror

class MerrorCls

Merror 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.niotMeas.multiEval.merror.clone()
```

Subgroups

6.2.1.5.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:MERRor:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:MERRor:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:MERRor:AVERage
value: List[float] = driver.niotMeas.multiEval.merror.average.fetch()
```

Returns the values of the magnitude error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:MERRor:AVERage
value: List[float] = driver.niotMeas.multiEval.merror.average.read()
```

Returns the values of the magnitude error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.5.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:MERRor:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:MERRor:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:MERRor:CURRent
value: List[float] = driver.niotMeas.multiEval.merror.current.fetch()
```

Returns the values of the magnitude error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:MERRor:CURRent
value: List[float] = driver.niotMeas.multiEval.merror.current.read()
```

Returns the values of the magnitude error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.5.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:MERRor:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:MERRor:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:MERRor:MAXimum
value: List[float] = driver.niotMeas.multiEval.merror.maximum.fetch()
```

Returns the values of the magnitude error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:MERRor:MAXimum
value: List[float] = driver.niotMeas.multiEval.merror.maximum.read()
```

Returns the values of the magnitude error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.6 Modulation

class ModulationCls

Modulation 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.niotMeas.multiEval.modulation.clone()
```

Subgroups

6.2.1.6.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:AVERage
CALCulate:NIOT:MEASurement<Instance>:MEValuation:MODulation:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms: float or bool: Error vector magnitude RMS value
- Evm_Peak: float or bool: Error vector magnitude peak value
- Mag_Error_Rms: float or bool: Magnitude error RMS value
- Mag_Err_Peak: float or bool: Magnitude error peak value
- Ph_Error_Rms: float or bool: Phase error RMS value
- Ph_Error_Peak: float or bool: Phase error peak value
- Iq_Offset: float or bool: I/Q origin offset
- Frequency_Error: float or bool: Carrier frequency error
- Timing_Error: float or bool: Transmit time error.
- Tx_Power: float or bool: User equipment power
- Peak_Power: float or bool: User equipment peak power
- Sc_Power: float or bool: Power in allocated subcarriers
- Evm_Dmrs: float or bool: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float or bool: Magnitude error DMRS value
- Ph_Error_Dmrs: float or bool: Phase error DMRS value
- Iq_Gain_Imbalance: float or bool: No parameter help available
- Iq_Quadrature_Err: float or bool: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms: float: Error vector magnitude RMS value

- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value
- Iq_Offset: float: I/Q origin offset
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error.
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power
- Sc_Power: float: Power in allocated subcarriers
- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value
- Ph_Error_Dmrs: float: Phase error DMRS value
- Iq_Gain_Imbalance: float: No parameter help available
- Iq_Quadrature_Err: float: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:MODulation:AVERage
value: CalculateStruct = driver.niotMeas.multiEval.modulation.average.
↪calculate()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:AVERage
value: ResultData = driver.niotMeas.multiEval.modulation.average.fetch()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:AVERage
value: ResultData = driver.niotMeas.multiEval.modulation.average.read()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.6.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:CURRent
CALCulate:NIOT:MEASurement<Instance>:MEValuation:MODulation:CURRent
```

`class CurrentCls`

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

`class CalculateStruct`

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms: float or bool: Error vector magnitude RMS value
- Evm_Peak: float or bool: Error vector magnitude peak value
- Mag_Error_Rms: float or bool: Magnitude error RMS value
- Mag_Err_Peak: float or bool: Magnitude error peak value
- Ph_Error_Rms: float or bool: Phase error RMS value
- Ph_Error_Peak: float or bool: Phase error peak value
- Iq_Offset: float or bool: I/Q origin offset
- Frequency_Error: float or bool: Carrier frequency error
- Timing_Error: float or bool: Transmit time error.
- Tx_Power: float or bool: User equipment power
- Peak_Power: float or bool: User equipment peak power
- Sc_Power: float or bool: Power in allocated subcarriers
- Evm_Dmrs: float or bool: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float or bool: Magnitude error DMRS value
- Ph_Error_Dmrs: float or bool: Phase error DMRS value
- Iq_Gain_Imbalance: float or bool: No parameter help available
- Iq_Quadrature_Err: float or bool: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms: float: Error vector magnitude RMS value
- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value
- Iq_Offset: float: I/Q origin offset
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error.
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power
- Sc_Power: float: Power in allocated subcarriers
- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value
- Ph_Error_Dmrs: float: Phase error DMRS value
- Iq_Gain_Imbalance: float: No parameter help available
- Iq_Quadrature_Err: float: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:MODulation:CURRent
value: CalculateStruct = driver.niotMeas.multiEval.modulation.current.
↪calculate()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:CURRent
value: ResultData = driver.niotMeas.multiEval.modulation.current.fetch()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:CURRent
value: ResultData = driver.niotMeas.multiEval.modulation.current.read()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.6.3 Extreme

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:EXTReme
FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:EXTReme
CALCulate:NIOT:MEASurement<Instance>:MEValuation:MODulation:EXTReme
```

class ExtremeCls

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

class CalculateStruct

Response structure. Fields:

- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms: float or bool: Error vector magnitude RMS value
- Evm_Peak: float or bool: Error vector magnitude peak value
- Mag_Error_Rms: float or bool: Magnitude error RMS value
- Mag_Err_Peak: float or bool: Magnitude error peak value
- Ph_Error_Rms: float or bool: Phase error RMS value
- Ph_Error_Peak: float or bool: Phase error peak value
- Iq_Offset: float or bool: I/Q origin offset
- Frequency_Error: float or bool: Carrier frequency error
- Timing_Error: float or bool: Transmit time error
- Tx_Power_Minimum: float or bool: Minimum user equipment power
- Tx_Power_Maximum: float or bool: Maximum user equipment power
- Peak_Power_Min: float or bool: Minimum user equipment peak power
- Peak_Power_Max: float or bool: Maximum user equipment peak power
- Sc_Pow_Min: float or bool: No parameter help available
- Sc_Pow_Max: float or bool: No parameter help available

- Evm_Dmrs: float or bool: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float or bool: Magnitude error DMRS value
- Ph_Error_Dmrs: float or bool: Phase error DMRS value
- Iq_Gain_Imbalance: float or bool: No parameter help available
- Iq_Quadrature_Err: float or bool: No parameter help available

class ResultData

Response structure. Fields:

- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms: float: Error vector magnitude RMS value
- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value
- Iq_Offset: float: I/Q origin offset
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power_Minimum: float: Minimum user equipment power
- Tx_Power_Maximum: float: Maximum user equipment power
- Peak_Power_Min: float: Minimum user equipment peak power
- Peak_Power_Max: float: Maximum user equipment peak power
- Sc_Pow_Min: float: No parameter help available
- Sc_Pow_Max: float: No parameter help available
- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value
- Ph_Error_Dmrs: float: Phase error DMRS value
- Iq_Gain_Imbalance: float: No parameter help available
- Iq_Quadrature_Err: float: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:MODulation:EXTReme
value: CalculateStruct = driver.niotMeas.multiEval.modulation.extreme.
↪calculate()
```

Returns the extreme single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

Suppressed linked return values: reliability

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:EXTReme
value: ResultData = driver.niotMeas.multiEval.modulation.extreme.fetch()
```

Returns the extreme single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

Suppressed linked return values: reliability

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:EXTReme
value: ResultData = driver.niotMeas.multiEval.modulation.extreme.read()
```

Returns the extreme single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

Suppressed linked return values: reliability

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

6.2.1.6.4 StandardDev

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:SDEViation
FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:SDEViation
```

class StandardDevCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms: float: Error vector magnitude RMS value
- Evm_Peak: float: Error vector magnitude peak value
- Mag_Error_Rms: float: Magnitude error RMS value
- Mag_Err_Peak: float: Magnitude error peak value
- Ph_Error_Rms: float: Phase error RMS value
- Ph_Error_Peak: float: Phase error peak value
- Iq_Offset: float: I/Q origin offset

- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error.
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power
- Sc_Power: float: Power in allocated subcarriers
- Evm_Dmrs: float: Error vector magnitude DMRS value
- Mag_Err_Dmrs: float: Magnitude error DMRS value
- Ph_Error_Dmrs: float: Phase error DMRS value
- Iq_Gain_Imbalance: float: No parameter help available
- Iq_Quadrature_Err: float: No parameter help available

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:MODulation:SDEViation
value: ResultData = driver.niotMeas.multiEval.modulation.standardDev.fetch()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:MODulation:SDEViation
value: ResultData = driver.niotMeas.multiEval.modulation.standardDev.read()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.7 Pdynamics

class PdynamicsCls

Pdynamics commands group definition. 14 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.1.7.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:AVERage
CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float or bool: No parameter help available
- On_Power_Rms: float or bool: No parameter help available
- On_Power_Peak: float or bool: No parameter help available
- Off_Power_After: float or bool: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: No parameter help available
- On_Power_Rms: float: No parameter help available
- On_Power_Peak: float: No parameter help available
- Off_Power_After: float: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:AVERage
value: CalculateStruct = driver.niotMeas.multiEval.pdynamics.average.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:AVERage
value: ResultData = driver.niotMeas.multiEval.pdynamics.average.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:AVERage
value: ResultData = driver.niotMeas.multiEval.pdynamics.average.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.7.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:CURRent
CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float or bool: No parameter help available
- On_Power_Rms: float or bool: No parameter help available
- On_Power_Peak: float or bool: No parameter help available
- Off_Power_After: float or bool: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: No parameter help available
- On_Power_Rms: float: No parameter help available
- On_Power_Peak: float: No parameter help available

- Off_Power_After: float: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:CURRent
value: CalculateStruct = driver.niotMeas.multiEval.pdynamics.current.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:CURRent
value: ResultData = driver.niotMeas.multiEval.pdynamics.current.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:CURRent
value: ResultData = driver.niotMeas.multiEval.pdynamics.current.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.7.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MAXimum
CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MAXimum
```

class MaximumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.

- Off_Power_Before: float or bool: No parameter help available
- On_Power_Rms: float or bool: No parameter help available
- On_Power_Peak: float or bool: No parameter help available
- Off_Power_After: float or bool: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: No parameter help available
- On_Power_Rms: float: No parameter help available
- On_Power_Peak: float: No parameter help available
- Off_Power_After: float: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MAXimum
value: CalculateStruct = driver.niotMeas.multiEval.pdynamics.maximum.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MAXimum
value: ResultData = driver.niotMeas.multiEval.pdynamics.maximum.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MAXimum
value: ResultData = driver.niotMeas.multiEval.pdynamics.maximum.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.7.4 Minimum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MINimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MINimum
CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MINimum
```

class MinimumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float or bool: No parameter help available
- On_Power_Rms: float or bool: No parameter help available
- On_Power_Peak: float or bool: No parameter help available
- Off_Power_After: float or bool: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: No parameter help available
- On_Power_Rms: float: No parameter help available
- On_Power_Peak: float: No parameter help available
- Off_Power_After: float: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MINimum
value: CalculateStruct = driver.niotMeas.multiEval.pdynamics.minimum.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MINimum
value: ResultData = driver.niotMeas.multiEval.pdynamics.minimum.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:MINimum
value: ResultData = driver.niotMeas.multiEval.pdynamics.minimum.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.7.5 StandardDev

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:SDEViation
FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:SDEViation
```

class StandardDevCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: No parameter help available
- On_Power_Rms: float: No parameter help available
- On_Power_Peak: float: No parameter help available
- Off_Power_After: float: No parameter help available

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:SDEViation
value: ResultData = driver.niotMeas.multiEval.pdynamics.standardDev.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PDYNamics:SDEViation
value: ResultData = driver.niotMeas.multiEval.pdynamics.standardDev.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.8 Perror

class PerrorCls

Perror 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.niotMeas.multiEval.perror.clone()
```

Subgroups

6.2.1.8.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PERRor:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:PERRor:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PERRor:AVERage
value: List[float] = driver.niotMeas.multiEval.perror.average.fetch()
```

Returns the values of the phase error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PERRor:AVERage
value: List[float] = driver.niotMeas.multiEval.perror.average.read()
```

Returns the values of the phase error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.8.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PERRor:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:PERRor:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PERRor:CURRent
value: List[float] = driver.niotMeas.multiEval.perror.current.fetch()
```

Returns the values of the phase error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PERRor:CURRent
value: List[float] = driver.niotMeas.multiEval.perror.current.read()
```

Returns the values of the phase error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.8.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:PERRor:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:PERRor:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:PERRor:MAXimum
value: List[float] = driver.niotMeas.multiEval.perror.maximum.fetch()
```

Returns the values of the phase error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:PERRor:MAXimum
value: List[float] = driver.niotMeas.multiEval.perror.maximum.read()
```

Returns the values of the phase error bar graphs for the SC-FDMA symbols in the measured slot. The results of the current, average and maximum bar graphs can be retrieved. See also 'Squares Magnitute Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 7 results, for SC-FDMA symbol 0 to 6

6.2.1.9 SeMask

class SeMaskCls

SeMask commands group definition. 18 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.1.9.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:AVERage
CALCulate:NIOT:MEASurement<Instance>:MEValuation:SEMask:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Obw: enums.ResultStatus2: Occupied bandwidth
- Tx_Power: enums.ResultStatus2: Total TX power within 400 kHz

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Obw: float: Occupied bandwidth
- Tx_Power: float: Total TX power within 400 kHz

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:SEMask:AVERage
value: CalculateStruct = driver.niotMeas.multiEval.seMask.average.calculate()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:AVERage
value: ResultData = driver.niotMeas.multiEval.seMask.average.fetch()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:AVERage
value: ResultData = driver.niotMeas.multiEval.seMask.average.read()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.9.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:CURRent
CALCulate:NIOT:MEASurement<Instance>:MEValuation:SEMask:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Obw: enums.ResultStatus2: Occupied bandwidth
- Tx_Power: enums.ResultStatus2: Total TX power within 400 kHz

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Obw: float: Occupied bandwidth
- Tx_Power: float: Total TX power within 400 kHz

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:SEMask:CURRent
value: CalculateStruct = driver.niotMeas.multiEval.seMask.current.calculate()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:CURRent
value: ResultData = driver.niotMeas.multiEval.seMask.current.fetch()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:CURRent
value: ResultData = driver.niotMeas.multiEval.seMask.current.read()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.9.3 Extreme

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:EXTReme
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:EXTReme
CALCulate:NIOT:MEASurement<Instance>:MEValuation:SEMask:EXTReme
```

class ExtremeCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Obw: enums.ResultStatus2: Occupied bandwidth
- Tx_Power_Min: enums.ResultStatus2: Minimum total TX power
- Tx_Power_Max: enums.ResultStatus2: Maximum total TX power

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Obw: float: Occupied bandwidth

- Tx_Power_Min: float: Minimum total TX power
- Tx_Power_Max: float: Maximum total TX power

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:MEValuation:SEMask:EXTReme
value: CalculateStruct = driver.niotMeas.multiEval.seMask.extreme.calculate()
```

Return the extreme single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:EXTReme
value: ResultData = driver.niotMeas.multiEval.seMask.extreme.fetch()
```

Return the extreme single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:EXTReme
value: ResultData = driver.niotMeas.multiEval.seMask.extreme.read()
```

Return the extreme single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.9.4 Margin

class MarginCls

Margin commands group definition. 7 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.1.9.4.1 All

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:ALL
```

class AllCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Margin_Curr_Neg: List[float]: Comma-separated list of 12 margin results For the current trace, area 1 to 12, negative frequency offsets
- Margin_Curr_Pos: List[float]: Comma-separated list of 12 margin results For the current trace, area 1 to 12, positive frequency offsets
- Margin_Avg_Neg: List[float]: Comma-separated list of 12 margin results For the average trace, area 1 to 12, negative frequency offsets
- Margin_Avg_Pos: List[float]: Comma-separated list of 12 margin results For the average trace, area 1 to 12, positive frequency offsets
- Margin_Min_Neg: List[float]: Comma-separated list of 12 margin results For the maximum trace, area 1 to 12, negative frequency offsets
- Margin_Min_Pos: List[float]: Comma-separated list of 12 margin results For the maximum trace, area 1 to 12, positive frequency offsets

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:ALL
value: FetchStruct = driver.niotMeas.multiEval.seMask.margin.all.fetch()
```

Returns spectrum emission mask margin results. A negative margin indicates that the trace is located above the limit line, i.e. the limit is exceeded. Results are provided for the current, average and maximum traces. For inactive areas, NCAP is returned.

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

6.2.1.9.4.2 Average

class AverageCls

Average 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.niotMeas.multiEval.seMask.margin.average.clone()
```

Subgroups

6.2.1.9.4.3 Negativ

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:AVERage:NEGativ
```

class NegativCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Margin_Avg_Neg_X: List[float]: No parameter help available
- Margin_Avg_Neg_Y: List[float]: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:SEMask:MARGin:AVERage:NEGativ
value: FetchStruct = driver.niotMeas.multiEval.seMask.margin.average.negativ.
↪fetch()
```

Returns spectrum emission mask margin results. A negative margin indicates that the trace is located above the limit line, i.e. the limit is exceeded. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) , for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {...}area 12 For inactive areas, NCAP is returned.

return

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

6.2.1.9.4.4 Positiv

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:AVERage:POSitiv
```

class PositivCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Margin_Avg_Pos_X: List[float]: X-position of margin
- Margin_Avg_Pos_Y: List[float]: Y-value of margin

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:SEMask:MARGin:AVERage:POSitiv
value: FetchStruct = driver.niotMeas.multiEval.seMask.margin.average.positiv.
↪fetch()
```

Returns spectrum emission mask margin results. A negative margin indicates that the trace is located above the limit line, i.e. the limit is exceeded. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) , for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {...}area 12 For inactive areas, NCAP is returned.

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

6.2.1.9.4.5 Current

class CurrentCls

Current 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.niotMeas.multiEval.seMask.margin.current.clone()
```

Subgroups

6.2.1.9.4.6 Negativ

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:CURRent:NEGativ
```

class NegativCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Margin_Curr_Neg_X: List[float]: No parameter help available
- Margin_Curr_Neg_Y: List[float]: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:SEMask:MARGin:CURRent:NEGativ
value: FetchStruct = driver.niotMeas.multiEval.seMask.margin.current.negativ.
↪fetch()
```

Returns spectrum emission mask margin results. A negative margin indicates that the trace is located above the limit line, i.e. the limit is exceeded. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) , for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {...}area 12 For inactive areas, NCAP is returned.

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

6.2.1.9.4.7 Positiv

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:CURRent:POSitiv
```

class PositivCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.

- Margin_Curr_Pos_X: List[float]: X-position of margin
- Margin_Curr_Pos_Y: List[float]: Y-value of margin

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:SEMask:MARGin:CURRent:POSitiv
value: FetchStruct = driver.niotMeas.multiEval.seMask.margin.current.positiv.
↪fetch()
```

Returns spectrum emission mask margin results. A negative margin indicates that the trace is located above the limit line, i.e. the limit is exceeded. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) , for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {...}area 12 For inactive areas, NCAP is returned.

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

6.2.1.9.4.8 Minimum

class MinimumCls

Minimum 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.niotMeas.multiEval.seMask.margin.minimum.clone()
```

Subgroups

6.2.1.9.4.9 Negativ

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:MINimum:NEGativ
```

class NegativCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Margin_Min_Neg_X: List[float]: No parameter help available
- Margin_Min_Neg_Y: List[float]: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:SEMask:MARGin:MINimum:NEGativ
value: FetchStruct = driver.niotMeas.multiEval.seMask.margin.minimum.negativ.
↪fetch()
```

Returns spectrum emission mask margin results. A negative margin indicates that the trace is located above the limit line, i.e. the limit is exceeded. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) , for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {...}area 12 For inactive areas, NCAP is returned.

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

6.2.1.9.4.10 Positiv

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:MARGin:MINimum:POSitiv
```

class PositivCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Margin_Min_Pos_X: List[float]: X-position of margin
- Margin_Min_Pos_Y: List[float]: Y-value of margin

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:SEMask:MARGin:MINimum:POSitiv
value: FetchStruct = driver.niotMeas.multiEval.seMask.margin.minimum.positiv.
↪fetch()
```

Returns spectrum emission mask margin results. A negative margin indicates that the trace is located above the limit line, i.e. the limit is exceeded. The individual commands provide results for the CURRent, AVERage and maximum traces (resulting in MINimum margins) , for NEGative and POSitive offset frequencies. Each command returns pairs of X and Y values of the margins for emission mask areas 1 to 12. Result array structure: <Reliability>, <OutOfTolerance>, {<MarginX>, <MarginY>}area 1, {...}area 2, ..., {...}area 12 For inactive areas, NCAP is returned.

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

6.2.1.9.5 StandardDev

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:SDEViation
FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:SDEViation
```

class StandardDevCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for spectrum emission measurements exceeding the specified spectrum emission mask limits.
- Obw: float: Occupied bandwidth
- Tx_Power: float: Total TX power within 400 kHz

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:SEMask:SDEViation
value: ResultData = driver.niotMeas.multiEval.seMask.standardDev.fetch()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:SEMask:SDEViation
value: ResultData = driver.niotMeas.multiEval.seMask.standardDev.read()
```

Return the current, average and standard deviation single value results of the spectrum emission measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.1.10 State

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:STATe
value: enums.ResourceState = driver.niotMeas.multiEval.state.fetch(timeout = 1.
↪0, target_main_state = enums.TargetStateA.OFF, target_sync_state = enums.
↪TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_status: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

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

Subgroups

6.2.1.10.1 All

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:STATe:ALL
value: List[enums.ResourceState] = driver.niotMeas.multiEval.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
state: No help available

6.2.1.11 Trace

class TraceCls

Trace commands group definition. 33 total commands, 7 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.1.11.1 Aclr

class AclrCls

Aclr 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.niotMeas.multiEval.trace.aclr.clone()
```

Subgroups

6.2.1.11.1.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:AVERage
```

class AverageCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Utra_Neg: float: Power in the adjacent UTRA channel with lower frequency
- Gsm_Neg: float: Power in the adjacent GSM channel with lower frequency
- Nb_Iot: float: Power in the allocated NB-IoT channel
- Gsm_Pos: float: Power in the adjacent GSM channel with higher frequency
- Utra_Pos: float: Power in the adjacent UTRA channel with higher frequency

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:AVERage
value: ResultData = driver.niotMeas.multiEval.trace.aclr.average.fetch()
```

Returns the absolute powers as displayed in the ACLR diagram. The current and average values can be retrieved.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:AVERage
value: ResultData = driver.niotMeas.multiEval.trace.aclr.average.read()
```

Returns the absolute powers as displayed in the ACLR diagram. The current and average values can be retrieved.

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

6.2.1.11.1.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:CURRent
```

class CurrentCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Utra_Neg: float: Power in the adjacent UTRA channel with lower frequency
- Gsm_Neg: float: Power in the adjacent GSM channel with lower frequency
- Nb_Iot: float: Power in the allocated NB-IoT channel
- Gsm_Pos: float: Power in the adjacent GSM channel with higher frequency

- Utra_Pos: float: Power in the adjacent UTRA channel with higher frequency

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:CURRent
value: ResultData = driver.niotMeas.multiEval.trace.aclr.current.fetch()
```

Returns the absolute powers as displayed in the ACLR diagram. The current and average values can be retrieved.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:ACLR:CURRent
value: ResultData = driver.niotMeas.multiEval.trace.aclr.current.read()
```

Returns the absolute powers as displayed in the ACLR diagram. The current and average values can be retrieved.

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

6.2.1.11.2 EvmSymbol

class EvmSymbolCls

EvmSymbol 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.niotMeas.multiEval.trace.evmSymbol.clone()
```

Subgroups

6.2.1.11.2.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.evmSymbol.average.fetch()
```

Returns the values of the EVM vs modulation symbol trace. See also 'Square EVM'.

Suppressed linked return values: reliability

return
ratio: Comma-separated list of EVM values, one value per modulation symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.evmSymbol.average.read()
```

Returns the values of the EVM vs modulation symbol trace. See also 'Square EVM'.

Suppressed linked return values: reliability

return
ratio: Comma-separated list of EVM values, one value per modulation symbol

6.2.1.11.2.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.evmSymbol.current.fetch()
```

Returns the values of the EVM vs modulation symbol trace. See also 'Square EVM'.

Suppressed linked return values: reliability

return
ratio: Comma-separated list of EVM values, one value per modulation symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.evmSymbol.current.read()
```

Returns the values of the EVM vs modulation symbol trace. See also 'Square EVM'.

Suppressed linked return values: reliability

return
ratio: Comma-separated list of EVM values, one value per modulation symbol

6.2.1.11.2.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.evmSymbol.maximum.fetch()
```

Returns the values of the EVM vs modulation symbol trace. See also 'Square EVM'.

Suppressed linked return values: reliability

return
ratio: Comma-separated list of EVM values, one value per modulation symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:EVMSymbol:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.evmSymbol.maximum.read()
```

Returns the values of the EVM vs modulation symbol trace. See also 'Square EVM'.

Suppressed linked return values: reliability

return
ratio: Comma-separated list of EVM values, one value per modulation symbol

6.2.1.11.3 Iemissions

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:IEMissions
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:IEMissions
```

class IemissionsCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:IEMissions
value: List[float] = driver.niotMeas.multiEval.trace.iemissions.fetch()
```

Returns the values of the inband emissions trace. See also 'Square Inband Emissions'.

Suppressed linked return values: reliability

return
power: Comma-separated list of power values, one value per subcarrier For 15 kHz

SC spacing, 12 power values are returned. For 3.75 kHz SC spacing, 48 power values are returned.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:IEMissions
value: List[float] = driver.niotMeas.multiEval.trace.iemissions.read()
```

Returns the values of the inband emissions trace. See also 'Square Inband Emissions'.

Suppressed linked return values: reliability

return
power: Comma-separated list of power values, one value per subcarrier For 15 kHz SC spacing, 12 power values are returned. For 3.75 kHz SC spacing, 48 power values are returned.

6.2.1.11.4 Iq

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:IQ
```

class IqCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Iphase: List[float]: Normalized I amplitude
- Qphase: List[float]: Normalized Q amplitude

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:IQ
value: FetchStruct = driver.niotMeas.multiEval.trace.iq.fetch()
```

Returns the results in the I/Q constellation diagram as pairs of I/Q values: <Reliability>, {<IPhase>, <QPhase>}point 1, {...}point 2, ..., {...}point n See also 'Square IQ'

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

6.2.1.11.5 Pdynamics

class PdynamicsCls

Pdynamics 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.niotMeas.multiEval.trace.pdynamics.clone()
```

Subgroups

6.2.1.11.5.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.average.fetch()
```

Return the values of the left power dynamics trace (start of first allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the start of the first allocated RU. The other details depend on the subcarrier spacing, see table.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.average.read()
```

Return the values of the left power dynamics trace (start of first allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the start of the first allocated RU. The other details depend on the subcarrier spacing, see table.

6.2.1.11.5.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.current.fetch()
```

Return the values of the left power dynamics trace (start of first allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the start of the first allocated RU. The other details depend on the subcarrier spacing, see table.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.current.read()
```

Return the values of the left power dynamics trace (start of first allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the start of the first allocated RU. The other details depend on the subcarrier spacing, see table.

6.2.1.11.5.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.maximum.fetch()
```

Return the values of the left power dynamics trace (start of first allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the start of the first allocated RU. The other details depend on the subcarrier spacing, see table.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.maximum.read()
```

Return the values of the left power dynamics trace (start of first allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the start of the first allocated RU. The other details depend on the subcarrier spacing, see table.

6.2.1.11.5.4 Post

class PostCls

Post 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.niotMeas.multiEval.trace.pdynamics.post.clone()
```

Subgroups

6.2.1.11.5.5 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:TRACe:PDYNamics:POST:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.post.average.
↪fetch()
```

Return the values of the right power dynamics trace (end of last allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the end of the last allocated RU. The other details depend on the subcarrier spacing, see table.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.post.average.
↪read()
```

Return the values of the right power dynamics trace (end of last allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the end of the last allocated RU. The other details depend on the subcarrier spacing, see table.

6.2.1.11.5.6 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:TRACe:PDYNamics:POST:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.post.current.
↪fetch()
```

Return the values of the right power dynamics trace (end of last allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the end of the last allocated RU. The other details depend on the subcarrier spacing, see table.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.post.current.
↪read()
```

Return the values of the right power dynamics trace (end of last allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the end of the last allocated RU. The other details depend on the subcarrier spacing, see table.

6.2.1.11.5.7 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>
↪:MEValuation:TRACe:PDYNamics:POST:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.post.maximum.
↪fetch()
```

Return the values of the right power dynamics trace (end of last allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the end of the last allocated RU. The other details depend on the subcarrier spacing, see table.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PDYNamics:POST:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.pdynamics.post.maximum.
↪read()
```

Return the values of the right power dynamics trace (end of last allocated RU) . The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1408 power values The 705th value refers to the time 0 µs, the end of the last allocated RU. The other details depend on the subcarrier spacing, see table.

6.2.1.11.6 Pmonitor

SCPI Commands :

```
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PMONitor
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PMONitor
```

class PmonitorCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:PMONitor
value: List[float] = driver.niotMeas.multiEval.trace.pmonitor.fetch()
```

Returns the power monitor results for all captured slots. The number of captured slots is configurable, see method RsCMPX_NiotMeas.Configure.NiotMeas.MultiEval.nslots.

Suppressed linked return values: reliability

return
power: Comma-separated list of power values, one value per captured slot

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:PMONitor
value: List[float] = driver.niotMeas.multiEval.trace.pmonitor.read()
```

Returns the power monitor results for all captured slots. The number of captured slots is configurable, see method RsCMPX_NiotMeas.Configure.NiotMeas.MultiEval.nslots.

Suppressed linked return values: reliability

return
power: Comma-separated list of power values, one value per captured slot

6.2.1.11.7 SeMask

class SeMaskCls

SeMask 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.niotMeas.multiEval.trace.seMask.clone()
```

Subgroups

6.2.1.11.7.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:AVERage
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.seMask.average.fetch()
```

Returns the values of the spectrum emission traces. The results of the current, average and maximum traces can be retrieved. See also 'Square Spectrum Emission Mask'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1067 power results The value in the middle of the result array corresponds to the center frequency. The test point separation between two results equals 3.75 kHz.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:AVERage
value: List[float] = driver.niotMeas.multiEval.trace.seMask.average.read()
```

Returns the values of the spectrum emission traces. The results of the current, average and maximum traces can be retrieved. See also 'Square Spectrum Emission Mask'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1067 power results The value in the middle of the result array corresponds to the center frequency. The test point separation between two results equals 3.75 kHz.

6.2.1.11.7.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:CURRent
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.seMask.current.fetch()
```

Returns the values of the spectrum emission traces. The results of the current, average and maximum traces can be retrieved. See also 'Square Spectrum Emission Mask'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1067 power results The value in the middle of the result array corresponds to the center frequency. The test point separation between two results equals 3.75 kHz.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:CURRent
value: List[float] = driver.niotMeas.multiEval.trace.seMask.current.read()
```

Returns the values of the spectrum emission traces. The results of the current, average and maximum traces can be retrieved. See also 'Square Spectrum Emission Mask'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1067 power results The value in the middle of the result array corresponds to the center frequency. The test point separation between two results equals 3.75 kHz.

6.2.1.11.7.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:MAXimum
FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.seMask.maximum.fetch()
```

Returns the values of the spectrum emission traces. The results of the current, average and maximum traces can be retrieved. See also 'Square Spectrum Emission Mask'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1067 power results The value in the middle of the result array corresponds to the center frequency. The test point separation between two results equals 3.75 kHz.

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:MEValuation:TRACe:SEMask:MAXimum
value: List[float] = driver.niotMeas.multiEval.trace.seMask.maximum.read()
```

Returns the values of the spectrum emission traces. The results of the current, average and maximum traces can be retrieved. See also 'Square Spectrum Emission Mask'.

Suppressed linked return values: reliability

return
power: Comma-separated list of 1067 power results The value in the middle of the result array corresponds to the center frequency. The test point separation between two results equals 3.75 kHz.

6.2.2 Prach

SCPI Commands :

```
INITiate:NIOT:MEASurement<Instance>:PRACh
STOP:NIOT:MEASurement<Instance>:PRACh
ABORt:NIOT:MEASurement<Instance>:PRACh
```

class PrachCls

Prach commands group definition. 63 total commands, 4 Subgroups, 3 group commands

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

```
# SCPI: ABORt:NIOT:MEASurement<Instance>:PRACh
driver.niotMeas.prach.abort()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

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

```
# SCPI: INITiate:NIOT:MEASurement<Instance>:PRACh
driver.niotMeas.prach.initiate()
```

(continues on next page)

(continued from previous page)

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

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

stop() → None

```
# SCPI: STOP:NIOT:MEASurement<Instance>:PRACh
driver.niotMeas.prach.stop()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

```
# SCPI: STOP:NIOT:MEASurement<Instance>:PRACh
driver.niotMeas.prach.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the RUN state.
    - STOP... halts the measurement immediately. The measurement enters the RDY␣
↪state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the␣
↪OFF state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as stop, but waits for the operation to complete before continuing further. Use the RsCMPX_NiotMeas.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.niotMeas.prach.clone()
```

Subgroups

6.2.2.1 Modulation

class ModulationCls

Modulation commands group definition. 15 total commands, 6 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.2.1.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:MODulation:AVERage
FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:AVERage
CALCulate:NIOT:MEASurement<Instance>:PRACh:MODulation:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms_Low: float or bool: EVM RMS value, low EVM window position
- Evm_Rms_High: float or bool: EVM RMS value, high EVM window position
- Evm_Peak_Low: float or bool: EVM peak value, low EVM window position
- Evm_Peak_High: float or bool: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float or bool: Magnitude error RMS value, low EVM window position

- Mag_Error_Rms_High: float or bool: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float or bool: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float or bool: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float or bool: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float or bool: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float or bool: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float or bool: Phase error peak value, high EVM window position
- Frequency_Error: float or bool: Carrier frequency error
- Timing_Error: float or bool: Transmit time error
- Tx_Power: float or bool: User equipment power
- Peak_Power: float or bool: User equipment peak power

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms_Low: float: EVM RMS value, low EVM window position
- Evm_Rms_High: float: EVM RMS value, high EVM window position
- Evm_Peak_Low: float: EVM peak value, low EVM window position
- Evm_Peak_High: float: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float: Magnitude error RMS value, low EVM window position
- Mag_Error_Rms_High: float: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float: Phase error peak value, high EVM window position
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:PRACh:MODulation:AVERage
value: CalculateStruct = driver.niotMeas.prach.modulation.average.calculate()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:AVERage
value: ResultData = driver.niotMeas.prach.modulation.average.fetch()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:MODulation:AVERage
value: ResultData = driver.niotMeas.prach.modulation.average.read()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.1.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:MODulation:CURRent
FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:CURRent
CALCulate:NIOT:MEASurement<Instance>:PRACh:MODulation:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms_Low: float or bool: EVM RMS value, low EVM window position
- Evm_Rms_High: float or bool: EVM RMS value, high EVM window position
- Evm_Peak_Low: float or bool: EVM peak value, low EVM window position
- Evm_Peak_High: float or bool: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float or bool: Magnitude error RMS value, low EVM window position

- Mag_Error_Rms_High: float or bool: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float or bool: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float or bool: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float or bool: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float or bool: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float or bool: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float or bool: Phase error peak value, high EVM window position
- Frequency_Error: float or bool: Carrier frequency error
- Timing_Error: float or bool: Transmit time error
- Tx_Power: float or bool: User equipment power
- Peak_Power: float or bool: User equipment peak power

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms_Low: float: EVM RMS value, low EVM window position
- Evm_Rms_High: float: EVM RMS value, high EVM window position
- Evm_Peak_Low: float: EVM peak value, low EVM window position
- Evm_Peak_High: float: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float: Magnitude error RMS value, low EVM window position
- Mag_Error_Rms_High: float: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float: Phase error peak value, high EVM window position
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:PRACh:MODulation:CURRent
value: CalculateStruct = driver.niotMeas.prach.modulation.current.calculate()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:CURRent
value: ResultData = driver.niotMeas.prach.modulation.current.fetch()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:MODulation:CURRent
value: ResultData = driver.niotMeas.prach.modulation.current.read()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.1.3 Extreme

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:MODulation:EXTReme
FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:EXTReme
CALCulate:NIOT:MEASurement<Instance>:PRACh:MODulation:EXTReme
```

class ExtremeCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms_Low: float or bool: EVM RMS value, low EVM window position
- Evm_Rms_High: float or bool: EVM RMS value, high EVM window position
- Evm_Peak_Low: float or bool: EVM peak value, low EVM window position
- Evm_Peak_High: float or bool: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float or bool: Magnitude error RMS value, low EVM window position

- Mag_Error_Rms_High: float or bool: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float or bool: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float or bool: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float or bool: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float or bool: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float or bool: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float or bool: Phase error peak value, high EVM window position
- Frequency_Error: float or bool: Carrier frequency error
- Timing_Error: float or bool: Transmit time error
- Tx_Power_Min: float or bool: Minimum user equipment power
- Tx_Power_Max: float or bool: Maximum user equipment power
- Peak_Power_Min: float or bool: Minimum user equipment peak power
- Peak_Power_Max: float or bool: Maximum user equipment peak power

`class ResultData`

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms_Low: float: EVM RMS value, low EVM window position
- Evm_Rms_High: float: EVM RMS value, high EVM window position
- Evm_Peak_Low: float: EVM peak value, low EVM window position
- Evm_Peak_High: float: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float: Magnitude error RMS value, low EVM window position
- Mag_Error_Rms_High: float: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float: Phase error peak value, high EVM window position
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power_Min: float: Minimum user equipment power
- Tx_Power_Max: float: Maximum user equipment power
- Peak_Power_Min: float: Minimum user equipment peak power
- Peak_Power_Max: float: Maximum user equipment peak power

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:PRACh:MODulation:EXTReme
value: CalculateStruct = driver.niotMeas.prach.modulation.extreme.calculate()
```

Returns the extreme single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:EXTReme
value: ResultData = driver.niotMeas.prach.modulation.extreme.fetch()
```

Returns the extreme single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:MODulation:EXTReme
value: ResultData = driver.niotMeas.prach.modulation.extreme.read()
```

Returns the extreme single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.1.4 Preamble<Preamble>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.niotMeas.prach.modulation.preamble.repcap_preamble_get()
driver.niotMeas.prach.modulation.preamble.repcap_preamble_set(repcap.Preamble.Nr1)
```

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:MODulation:PREamble<Number>
FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:PREamble<Number>
```

class PreambleCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'

- Preamble_Rel: int: Reliability indicator for the preamble
- Evm_Rms_Low: float: EVM RMS value, low EVM window position
- Evm_Rms_High: float: EVM RMS value, high EVM window position
- Evm_Peak_Low: float: EVM peak value, low EVM window position
- Evm_Peak_High: float: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float: Magnitude error RMS value, low EVM window position
- Mag_Error_Rms_High: float: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float: Phase error peak value, high EVM window position
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power

fetch(*preamble=Preamble.Default*) → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:PREamble<Number>
value: ResultData = driver.niotMeas.prach.modulation.preamble.fetch(preamble =␣
↪repcap.Preamble.Default)
```

Return the single value results of the EVM vs Preamble and Power vs Preamble squares, for a selected preamble. See also 'Square EVM vs Preamble, Power vs Preamble'.

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

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

read(*preamble=Preamble.Default*) → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:MODulation:PREamble<Number>
value: ResultData = driver.niotMeas.prach.modulation.preamble.read(preamble =␣
↪repcap.Preamble.Default)
```

Return the single value results of the EVM vs Preamble and Power vs Preamble squares, for a selected preamble. See also 'Square EVM vs Preamble, Power vs Preamble'.

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

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.niotMeas.prach.modulation.preamble.clone()
```

6.2.2.1.5 ScsGroup

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:SCSGroup
```

class ScsGroupCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Sub_Carr_1: int: Subcarrier number for the first symbol group
- Sub_Carr_2: int: Subcarrier number for the second symbol group
- Sub_Carr_3: int: Subcarrier number for the third symbol group
- Sub_Carr_4: int: Subcarrier number for the fourth symbol group

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:SCSGroup
value: FetchStruct = driver.niotMeas.prach.modulation.scsGroup.fetch()
```

Returns the numbers of the subcarriers used by the symbol groups of the preamble, for single-preamble measurements. See also 'NB-IoT preamble structure'.

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.niotMeas.prach.modulation.scsGroup.clone()
```

Subgroups

6.2.2.1.5.1 Preamble<Preamble>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.niotMeas.prach.modulation.scsGroup.preamble.repcap_preamble_get()
driver.niotMeas.prach.modulation.scsGroup.preamble.repcap_preamble_set(repcap.Preamble.
↪Nr1)
```

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:SCSGroup:PREamble<Number>
```

class PreambleCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Sub_Carr_1: int: Subcarrier number for the first symbol group
- Sub_Carr_2: int: Subcarrier number for the second symbol group
- Sub_Carr_3: int: Subcarrier number for the third symbol group
- Sub_Carr_4: int: Subcarrier number for the fourth symbol group

fetch(*preamble=Preamble.Default*) → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:SCSGroup:PREamble
↪<Number>
value: FetchStruct = driver.niotMeas.prach.modulation.scsGroup.preamble.
↪fetch(preamble = repcap.Preamble.Default)
```

Returns the numbers of the subcarriers used by the symbol groups of a selected preamble, for multi-preamble measurements. See also 'NB-IoT preamble structure'.

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

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.niotMeas.prach.modulation.scsGroup.preamble.clone()
```

6.2.2.1.6 StandardDev

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:MODulation:SDEViation
FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:SDEViation
```

class StandardDevCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for modulation measurements exceeding the specified modulation limits.
- Evm_Rms_Low: float: EVM RMS value, low EVM window position
- Evm_Rms_High: float: EVM RMS value, high EVM window position
- Evm_Peak_Low: float: EVM peak value, low EVM window position
- Evm_Peak_High: float: EVM peak value, high EVM window position
- Mag_Error_Rms_Low: float: Magnitude error RMS value, low EVM window position
- Mag_Error_Rms_High: float: Magnitude error RMS value, low EVM window position
- Mag_Err_Peak_Low: float: Magnitude error peak value, low EVM window position
- Mag_Err_Peak_High: float: Magnitude error peak value, high EVM window position
- Ph_Error_Rms_Low: float: Phase error RMS value, low EVM window position
- Ph_Error_Rms_High: float: Phase error RMS value, high EVM window position
- Ph_Error_Peak_Low: float: Phase error peak value, low EVM window position
- Ph_Error_Peak_High: float: Phase error peak value, high EVM window position
- Frequency_Error: float: Carrier frequency error
- Timing_Error: float: Transmit time error
- Tx_Power: float: User equipment power
- Peak_Power: float: User equipment peak power

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:MODulation:SDEViation
value: ResultData = driver.niotMeas.prach.modulation.standardDev.fetch()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:MODulation:SDEViation
value: ResultData = driver.niotMeas.prach.modulation.standardDev.read()
```

Return the current, average and standard deviation single value results. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.2 Pdynamics

class PdynamicsCls

Pdynamics commands group definition. 14 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.2.2.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:AVERage
FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:AVERage
CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:AVERage
```

class AverageCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float or bool: OFF power before the preamble
- On_Power_Rms: float or bool: ON power mean value
- On_Power_Peak: float or bool: ON power peak value
- Off_Power_After: float or bool: OFF power after the preamble

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: OFF power before the preamble
- On_Power_Rms: float: ON power mean value
- On_Power_Peak: float: ON power peak value
- Off_Power_After: float: OFF power after the preamble

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:AVERage
value: CalculateStruct = driver.niotMeas.prach.pdynamics.average.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:AVERage
value: ResultData = driver.niotMeas.prach.pdynamics.average.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:AVERage
value: ResultData = driver.niotMeas.prach.pdynamics.average.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.2.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:CURRent
FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:CURRent
CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:CURRent
```

class CurrentCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float or bool: OFF power before the preamble
- On_Power_Rms: float or bool: ON power mean value
- On_Power_Peak: float or bool: ON power peak value
- Off_Power_After: float or bool: OFF power after the preamble

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: OFF power before the preamble
- On_Power_Rms: float: ON power mean value
- On_Power_Peak: float: ON power peak value
- Off_Power_After: float: OFF power after the preamble

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:CURRent
value: CalculateStruct = driver.niotMeas.prach.pdynamics.current.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:CURRent
value: ResultData = driver.niotMeas.prach.pdynamics.current.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:CURRent
value: ResultData = driver.niotMeas.prach.pdynamics.current.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.2.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MAXimum
FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MAXimum
CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MAXimum
```

class MaximumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float or bool: OFF power before the preamble
- On_Power_Rms: float or bool: ON power mean value
- On_Power_Peak: float or bool: ON power peak value
- Off_Power_After: float or bool: OFF power after the preamble

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: OFF power before the preamble
- On_Power_Rms: float: ON power mean value
- On_Power_Peak: float: ON power peak value

- Off_Power_After: float: OFF power after the preamble

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MAXimum
value: CalculateStruct = driver.niotMeas.prach.pdynamics.maximum.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MAXimum
value: ResultData = driver.niotMeas.prach.pdynamics.maximum.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MAXimum
value: ResultData = driver.niotMeas.prach.pdynamics.maximum.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.2.4 Minimum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MINimum
FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MINimum
CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MINimum
```

class MinimumCls

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.

- Off_Power_Before: float or bool: OFF power before the preamble
- On_Power_Rms: float or bool: ON power mean value
- On_Power_Peak: float or bool: ON power peak value
- Off_Power_After: float or bool: OFF power after the preamble

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: OFF power before the preamble
- On_Power_Rms: float: ON power mean value
- On_Power_Peak: float: ON power peak value
- Off_Power_After: float: OFF power after the preamble

calculate() → CalculateStruct

```
# SCPI: CALCulate:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MINimum
value: CalculateStruct = driver.niotMeas.prach.pdynamics.minimum.calculate()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MINimum
value: ResultData = driver.niotMeas.prach.pdynamics.minimum.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:MINimum
value: ResultData = driver.niotMeas.prach.pdynamics.minimum.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.2.5 StandardDev

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:SDEViation
FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:SDEViation
```

class StandardDevCls

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

class ResultData

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Out_Of_Tolerance: int: Out of tolerance result, i.e. percentage of measurement intervals of the statistic count for power dynamics measurements exceeding the specified power dynamics limits.
- Off_Power_Before: float: OFF power before the preamble
- On_Power_Rms: float: ON power mean value
- On_Power_Peak: float: ON power peak value
- Off_Power_After: float: OFF power after the preamble

fetch() → ResultData

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:PDYNamics:SDEViation
value: ResultData = driver.niotMeas.prach.pdynamics.standardDev.fetch()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:PDYNamics:SDEViation
value: ResultData = driver.niotMeas.prach.pdynamics.standardDev.read()
```

Return the current, average, minimum, maximum and standard deviation single value results of the power dynamics measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

6.2.2.3 State

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:PRACh:STATe
```

class StateCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → ResourceState

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:STATe
value: enums.ResourceState = driver.niotMeas.prach.state.fetch(timeout = 1.0,␣
↪target_main_state = enums.TargetStateA.OFF, target_sync_state = enums.
↪TargetSyncState.ADJusted)
```

Queries the main measurement state. Without query parameters, the state is returned immediately. With query parameters, the state is returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
meas_status: Current state or target state of ongoing state transition OFF: measurement off RUN: measurement running RDY: measurement completed

Cloning the Group

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

Subgroups

6.2.2.3.1 All

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:PRACh:STATe:ALL
```

class AllCls

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

fetch(*timeout: float = None*, *target_main_state: TargetStateA = None*, *target_sync_state: TargetSyncState = None*) → List[ResourceState]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:STATe:ALL
value: List[enums.ResourceState] = driver.niotMeas.prach.state.all.
↪fetch(timeout = 1.0, target_main_state = enums.TargetStateA.OFF, target_sync_
↪state = enums.TargetSyncState.ADJusted)
```

Queries the main measurement state and the measurement substates. Without query parameters, the states are returned immediately. With query parameters, the states are returned when the <TargetMainState> and the <TargetSyncState> are reached or when the <Timeout> expires.

param timeout
No help available

param target_main_state
Target MainState for the query Default is RUN.

param target_sync_state
Target SyncState for the query Default is ADJ.

return
state: No help available

6.2.2.4 Trace

class TraceCls

Trace commands group definition. 29 total commands, 7 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.2.2.4.1 Evm

class EvmCls

Evm 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.niotMeas.prach.trace.evm.clone()
```

Subgroups

6.2.2.4.1.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:AVERage
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:AVERage
value: List[float] = driver.niotMeas.prach.trace.evm.average.fetch()
```

Return the values of the EVM vs symbol traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:AVERage
value: List[float] = driver.niotMeas.prach.trace.evm.average.read()
```

Return the values of the EVM vs symbol traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.1.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:CURRent
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:CURRent
value: List[float] = driver.niotMeas.prach.trace.evm.current.fetch()
```

Return the values of the EVM vs symbol traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:CURRent
value: List[float] = driver.niotMeas.prach.trace.evm.current.read()
```

Return the values of the EVM vs symbol traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.1.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:MAXimum
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:MAXimum
value: List[float] = driver.niotMeas.prach.trace.evm.maximum.fetch()
```

Return the values of the EVM vs symbol traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVM:MAXimum
value: List[float] = driver.niotMeas.prach.trace.evm.maximum.read()
```

Return the values of the EVM vs symbol traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.2 EvPreamble

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVPReamble
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVPReamble
```

class EvPreambleCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:EVPReamble
value: List[float] = driver.niotMeas.prach.trace.evPreamble.fetch()
```

Return the values of the EVM vs preamble traces. See also 'Square EVM vs Preamble, Power vs Preamble'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 32 EVM values, for preamble 1 to 32 (NCAP for not measured preambles)

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:EVPReamble
value: List[float] = driver.niotMeas.prach.trace.evPreamble.read()
```

Return the values of the EVM vs preamble traces. See also 'Square EVM vs Preamble, Power vs Preamble'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 32 EVM values, for preamble 1 to 32 (NCAP for not measured preambles)

6.2.2.4.3 Iq

SCPI Command :

```
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:IQ
```

class IqCls

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

class FetchStruct

Response structure. Fields:

- Reliability: int: 'Reliability indicator'
- Iphase: List[float]: Normalized I amplitude
- Qphase: List[float]: Normalized Q amplitude

fetch() → FetchStruct

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:IQ
value: FetchStruct = driver.niotMeas.prach.trace.iq.fetch()
```

Returns the results in the I/Q constellation diagram. The return order is: <Reliability>, <IPhase>1, <QPhase>1, <IPhase>2, <QPhase>2, ... See also 'Square IQ'.

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

6.2.2.4.4 Merror

class MerrorCls

Merror 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.niotMeas.prach.trace.merror.clone()
```

Subgroups

6.2.2.4.4.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:AVERage
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:AVERage
value: List[float] = driver.niotMeas.prach.trace.merror.average.fetch()
```

Return the values of the magnitude error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:AVERage
value: List[float] = driver.niotMeas.prach.trace.merror.average.read()
```

Return the values of the magnitude error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.4.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:CURRent
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:CURRent
value: List[float] = driver.niotMeas.prach.trace.merror.current.fetch()
```

Return the values of the magnitude error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:CURRent
value: List[float] = driver.niotMeas.prach.trace.merror.current.read()
```

Return the values of the magnitude error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.4.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:MAXimum
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:MAXimum
value: List[float] = driver.niotMeas.prach.trace.merror.maximum.fetch()
```

Return the values of the magnitude error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:MERRor:MAXimum
value: List[float] = driver.niotMeas.prach.trace.merror.maximum.read()
```

Return the values of the magnitude error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.5 Pdynamics

class PdynamicsCls

Pdynamics 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.niotMeas.prach.trace.pdynamics.clone()
```

Subgroups

6.2.2.4.5.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:AVERage
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:AVERage
value: List[float] = driver.niotMeas.prach.trace.pdynamics.average.fetch()
```

Return the values of the power dynamics traces. Each value is sampled with 96 Ts, corresponding to 3.125 µs. The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: 2816 power values, from -1200 µs to +7596.875 µs relative to the start of the preamble. The values have a spacing of 3.125 µs. The 385th value is at the start of the preamble (0 µs) .

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:AVERage
value: List[float] = driver.niotMeas.prach.trace.pdynamics.average.read()
```

Return the values of the power dynamics traces. Each value is sampled with 96 Ts, corresponding to 3.125 µs. The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: 2816 power values, from -1200 µs to +7596.875 µs relative to the start of the preamble. The values have a spacing of 3.125 µs. The 385th value is at the start of the preamble (0 µs) .

6.2.2.4.5.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:CURRent
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:CURRent
value: List[float] = driver.niotMeas.prach.trace.pdynamics.current.fetch()
```

Return the values of the power dynamics traces. Each value is sampled with 96 Ts, corresponding to 3.125 µs. The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: 2816 power values, from -1200 µs to +7596.875 µs relative to the start of the preamble. The values have a spacing of 3.125 µs. The 385th value is at the start of the preamble (0 µs) .

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:CURRent
value: List[float] = driver.niotMeas.prach.trace.pdynamics.current.read()
```

Return the values of the power dynamics traces. Each value is sampled with 96 Ts, corresponding to 3.125 µs. The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: 2816 power values, from -1200 µs to +7596.875 µs relative to the start of the preamble. The values have a spacing of 3.125 µs. The 385th value is at the start of the preamble (0 µs) .

6.2.2.4.5.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:MAXimum
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:MAXimum
value: List[float] = driver.niotMeas.prach.trace.pdynamics.maximum.fetch()
```

Return the values of the power dynamics traces. Each value is sampled with 96 Ts, corresponding to 3.125 µs. The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: 2816 power values, from -1200 µs to +7596.875 µs relative to the start of the preamble. The values have a spacing of 3.125 µs. The 385th value is at the start of the preamble (0 µs) .

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PDYNamics:MAXimum
value: List[float] = driver.niotMeas.prach.trace.pdynamics.maximum.read()
```

Return the values of the power dynamics traces. Each value is sampled with 96 Ts, corresponding to 3.125 µs. The results of the current, average and maximum traces can be retrieved. See also 'Square Power Dynamics'.

Suppressed linked return values: reliability

return
power: 2816 power values, from -1200 µs to +7596.875 µs relative to the start of the preamble. The values have a spacing of 3.125 µs. The 385th value is at the start of the preamble (0 µs) .

6.2.2.4.6 Perror

class PerrorCls

Perror 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.niotMeas.prach.trace.perror.clone()
```

Subgroups

6.2.2.4.6.1 Average

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:AVERage
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:AVERage
```

class AverageCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:AVERage
value: List[float] = driver.niotMeas.prach.trace.perror.average.fetch()
```

Return the values of the phase error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:AVERage
value: List[float] = driver.niotMeas.prach.trace.perror.average.read()
```

Return the values of the phase error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.6.2 Current

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:CURRent
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:CURRent
```

class CurrentCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:CURRent
value: List[float] = driver.niotMeas.prach.trace.perror.current.fetch()
```

Return the values of the phase error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:CURRent
value: List[float] = driver.niotMeas.prach.trace.perror.current.read()
```

Return the values of the phase error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.6.3 Maximum

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:MAXimum
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:MAXimum
```

class MaximumCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:MAXimum
value: List[float] = driver.niotMeas.prach.trace.perror.maximum.fetch()
```

Return the values of the phase error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PERRor:MAXimum
value: List[float] = driver.niotMeas.prach.trace.perror.maximum.read()
```

Return the values of the phase error traces. Each value is averaged over the samples in one preamble symbol. The results of the current, average and maximum traces can be retrieved. See also 'Squares EVM, Magnitude Error, Phase Error'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 20 results, one result per symbol

6.2.2.4.7 PvPreamble

SCPI Commands :

```
READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PVPReamble
FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PVPReamble
```

class PvPreambleCls

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

fetch() → List[float]

```
# SCPI: FETCh:NIOT:MEASurement<Instance>:PRACh:TRACe:PVPReamble
value: List[float] = driver.niotMeas.prach.trace.pvPreamble.fetch()
```

Return the values of the power vs preamble traces. See also 'Square EVM vs Preamble, Power vs Preamble'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 32 power values, for preamble 1 to 32 (NCAP for not measured preambles)

read() → List[float]

```
# SCPI: READ:NIOT:MEASurement<Instance>:PRACh:TRACe:PVPReamble
value: List[float] = driver.niotMeas.prach.trace.pvPreamble.read()
```

Return the values of the power vs preamble traces. See also 'Square EVM vs Preamble, Power vs Preamble'.

Suppressed linked return values: reliability

return
results: Comma-separated list of 32 power values, for preamble 1 to 32 (NCAP for not measured preambles)

6.3 Sense

class SenseCls

Sense 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.clone()
```

Subgroups

6.3.1 NiotMeas

class NiotMeasCls

NiotMeas 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.niotMeas.clone()
```

Subgroups

6.3.1.1 MultiEval

SCPI Command :

```
SENSe:NIOT:MEASurement<Instance>:MEValuation:NSRunits
```

class MultiEvalCls

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

get_nsr_units() → int

```
# SCPI: SENSe:NIOT:MEASurement<Instance>:MEValuation:NSRunits
value: int = driver.sense.niotMeas.multiEval.get_nsr_units()
```

Queries the number of slots per resource unit.

return
nof_slots: No help available

6.4 Trigger

class TriggerCls

Trigger commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.1 NiotMeas

class NiotMeasCls

NiotMeas commands group definition. 10 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.4.1.1 MultiEval

SCPI Commands :

```
TRIGger:NIOT:MEASurement<Instance>:MEValuation:THReshold
TRIGger:NIOT:MEASurement<Instance>:MEValuation:SLOPe
TRIGger:NIOT:MEASurement<Instance>:MEValuation:DELay
TRIGger:NIOT:MEASurement<Instance>:MEValuation:TOUT
TRIGger:NIOT:MEASurement<Instance>:MEValuation:MGAP
```

class MultiEvalCls

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

get_delay() → float

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:DELay
value: float = driver.trigger.niotMeas.multiEval.get_delay()
```

Defines a time delaying the start of the measurement relative to the trigger event. This setting has no influence on free run measurements.

return
delay: No help available

get_mgap() → float

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:MGAP
value: float = driver.trigger.niotMeas.multiEval.get_mgap()
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

return
min_trig_gap: No help available

get_slope() → SignalSlope

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:SLOPe
value: enums.SignalSlope = driver.trigger.niotMeas.multiEval.get_slope()
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

return
slope: REDGe: Rising edge FEDGe: Falling edge

get_threshold() → float

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:THReshold
value: float or bool = driver.trigger.niotMeas.multiEval.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
trig_threshold: (float or boolean) No help available

get_timeout() → float

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:TOUT
value: float or bool = driver.trigger.niotMeas.multiEval.get_timeout()
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

return
trigger_timeout: (float or boolean) No help available

set_delay(*delay: float*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:DELay
driver.trigger.niotMeas.multiEval.set_delay(delay = 1.0)
```

Defines a time delaying the start of the measurement relative to the trigger event. This setting has no influence on free run measurements.

param delay
No help available

set_mgap(*min_trig_gap: float*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:MGAP
driver.trigger.niotMeas.multiEval.set_mgap(min_trig_gap = 1.0)
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

param min_trig_gap
No help available

set_slope(*slope: SignalSlope*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:SLOPe
driver.trigger.niotMeas.multiEval.set_slope(slope = enums.SignalSlope.FEDGe)
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

param slope
REDGe: Rising edge FEDGe: Falling edge

set_threshold(*trig_threshold: float*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:THReshold
driver.trigger.niotMeas.multiEval.set_threshold(trig_threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param trig_threshold
(float or boolean) No help available

set_timeout(*trigger_timeout: float*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:TOUT
driver.trigger.niotMeas.multiEval.set_timeout(trigger_timeout = 1.0)
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

param trigger_timeout
(float or boolean) No help available

Cloning the Group

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

Subgroups

6.4.1.1.1 ListPy

SCPI Command :

```
TRIGger:NIOT:MEASurement<Instance>:MEValuation:LIST:MODE
```

class ListPyCls

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

get_mode() → ListMode

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:LIST:MODE
value: enums.ListMode = driver.trigger.niotMeas.multiEval.listPy.get_mode()
```

Specifies the trigger mode for list mode measurements. For configuration of retrigger flags, see method RsCMPX_NiotMeas. Configure.NiotMeas.MultiEval.ListPy.Segment.Setup.set.

return

mode: ONCE: A trigger event is only required to start the measurement. As a result, the entire range of segments to be measured is captured without additional trigger event. The retrigger flag of the first segment is evaluated. The other retrigger flags are ignored. SEGMent: The retrigger flag of each segment is evaluated. It defines whether the measurement waits for a trigger event before capturing the segment, or not.

set_mode(*mode: ListMode*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:MEValuation:LIST:MODE
driver.trigger.niotMeas.multiEval.listPy.set_mode(mode = enums.ListMode.ONCE)
```

Specifies the trigger mode for list mode measurements. For configuration of retrigger flags, see method RsCMPX_NiotMeas. Configure.NiotMeas.MultiEval.ListPy.Segment.Setup.set.

param mode

ONCE: A trigger event is only required to start the measurement. As a result, the entire range of segments to be measured is captured without additional trigger event. The retrigger flag of the first segment is evaluated. The other retrigger flags are ignored. SEGMent: The retrigger flag of each segment is evaluated. It defines whether the measurement waits for a trigger event before capturing the segment, or not.

6.4.1.2 Prach

SCPI Commands :

```
TRIGger:NIOT:MEASurement<Instance>:PRACh:THReshold
TRIGger:NIOT:MEASurement<Instance>:PRACh:SLOPe
TRIGger:NIOT:MEASurement<Instance>:PRACh:TOUT
TRIGger:NIOT:MEASurement<Instance>:PRACh:MGAP
```

class PrachCls

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

get_mgap() → float

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:MGAP
value: float = driver.trigger.niotMeas.prach.get_mgap()
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

return
min_trig_gap: No help available

get_slope() → SignalSlope

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:SLOPe
value: enums.SignalSlope = driver.trigger.niotMeas.prach.get_slope()
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

return
slope: REDGe: Rising edge FEDGe: Falling edge

get_threshold() → float

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:THReshold
value: float or bool = driver.trigger.niotMeas.prach.get_threshold()
```

Defines the trigger threshold for power trigger sources.

return
trig_threshold: (float or boolean) No help available

get_timeout() → float

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:TOUT
value: float or bool = driver.trigger.niotMeas.prach.get_timeout()
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

return
trigger_timeout: (float or boolean) No help available

set_mgap(*min_trig_gap: float*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:MGAP
driver.trigger.niotMeas.prach.set_mgap(min_trig_gap = 1.0)
```

Sets a minimum time during which the IF signal must be below the trigger threshold before the trigger is armed so that an IF power trigger event can be generated.

param min_trig_gap
No help available

set_slope(*slope: SignalSlope*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:SLOPe
driver.trigger.niotMeas.prach.set_slope(slope = enums.SignalSlope.FEDGe)
```

Qualifies whether the trigger event is generated at the rising or at the falling edge of the trigger pulse (valid for external and power trigger sources) .

param slope
REDGe: Rising edge FEDGe: Falling edge

set_threshold(*trig_threshold: float*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:THReshold
driver.trigger.niotMeas.prach.set_threshold(trig_threshold = 1.0)
```

Defines the trigger threshold for power trigger sources.

param trig_threshold
(float or boolean) No help available

set_timeout(*trigger_timeout: float*) → None

```
# SCPI: TRIGger:NIOT:MEASurement<Instance>:PRACh:TOUT
driver.trigger.niotMeas.prach.set_timeout(trigger_timeout = 1.0)
```

Selects the maximum time that the measurement waits for a trigger event before it stops in remote control mode or indicates a trigger timeout in manual operation mode. This setting has no influence on Free Run measurements.

param trigger_timeout
(float or boolean) No help available

CHAPTER

SEVEN

RSCMPX_NIOTMEAS UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsCMPX_NiotMeas.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsCMPX_NiotMeas.utilities.logger`

property driver_version: str

Returns the instrument driver version.

property idn_string: str

Returns instrument's identification string - the response on the SCPI command *IDN?

property manufacturer: str

Returns manufacturer of the instrument.

property full_instrument_model_name: str

Returns the current instrument's full name e.g. 'FSW26'.

property instrument_model_name: str

Returns the current instrument's family name e.g. 'FSW'.

property supported_models: List[str]

Returns a list of the instrument models supported by this instrument driver.

property instrument_firmware_version: str

Returns instrument's firmware version.

property instrument_serial_number: str

Returns instrument's serial_number.

query_opc(*timeout: int = 0*) → int

SCPI command: *OPC? Queries the instrument's OPC bit and hence it waits until the instrument reports operation complete. If you define timeout > 0, the VISA timeout is set to that value just for this method call.

property instrument_status_checking: bool

Sets / returns Instrument Status Checking. When True (default is True), all the driver methods and properties are sending "SYSTem:ERRor?" at the end to immediately react on error that might have occurred. We recommend to keep the state checking ON all the time. Switch it OFF only in rare cases when you require maximum speed. The default state after initializing the session is ON.

property encoding: str

Returns string<=>bytes encoding of the session.

property opc_query_after_write: bool

Sets / returns Instrument *OPC? query sending after each command write. When True, (default is False) the driver sends *OPC? every time a write command is performed. Use this if you want to make sure your sequence is performed command-after-command.

property bin_float_numbers_format: BinFloatFormat

Sets / returns format of float numbers when transferred as binary data.

property bin_int_numbers_format: BinIntFormat

Sets / returns format of integer numbers when transferred as binary data.

clear_status() → None

Clears instrument's status system, the session's I/O buffers and the instrument's error queue.

query_all_errors() → List[str]

Queries and clears all the errors from the instrument's error queue. The method returns list of strings as error messages. If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty. If you want to include the error codes, call the query_all_errors_with_codes()

query_all_errors_with_codes() → List[Tuple[int, str]]

Queries and clears all the errors from the instrument's error queue. The method returns list of tuples (code: int, message: str). If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty.

property instrument_options: List[str]

Returns all the instrument options. The options are sorted in the ascending order starting with K-options and continuing with B-options.

reset() → None

SCPI command: *RST Sends *RST command + calls the clear_status().

default_instrument_setup() → None

Custom steps performed at the init and at the reset().

self_test(*timeout: int = None*) → Tuple[int, str]

SCPI command: *TST? Performs instrument's self-test. Returns tuple (code:int, message: str). Code 0 means the self-test passed. You can define the custom timeout in milliseconds. If you do not define it, the default selftest timeout is used (usually 60 secs).

is_connection_active() → bool

Returns true, if the VISA connection is active and the communication with the instrument still works.

reconnect(*force_close: bool = False*) → bool

If the connection is not active, the method tries to reconnect to the device If the connection is active, and force_close is False, the method does nothing. If the connection is active, and force_close is True, the method closes, and opens the session again. Returns True, if the reconnection has been performed.

property resource_name: int

Returns the resource name used in the constructor

property opc_timeout: int

Sets / returns timeout in milliseconds for all the operations that use OPC synchronization.

property visa_timeout: int

Sets / returns visa IO timeout in milliseconds.

property data_chunk_size: int

Sets / returns the maximum size of one block transferred during write/read operations

property visa_manufacturer: int

Returns the manufacturer of the current VISA session.

process_all_commands() → None

SCPI command: *WAI Stops further commands processing until all commands sent before *WAI have been executed.

write_str(*cmd: str*) → None

Writes the command to the instrument.

write(*cmd: str*) → None

This method is an alias to the write_str(). Writes the command to the instrument as string.

write_int(*cmd: str*, *param: int*) → None

Writes the command to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2'

write_int_with_opc(*cmd: str*, *param: int*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2' If you do not provide timeout, the method uses current opc_timeout.

write_float(*cmd: str*, *param: float*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6'

write_float_with_opc(*cmd: str*, *param: float*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6' If you do not provide timeout, the method uses current opc_timeout.

write_bool(*cmd: str*, *param: bool*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON'

write_bool_with_opc(*cmd: str*, *param: bool*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON' If you do not provide timeout, the method uses current opc_timeout.

query_str(*query: str*) → str

Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query(*query: str*) → str

This method is an alias to the query_str(). Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query_bool(*query: str*) → bool

Sends the query to the instrument and returns the response as boolean.

query_int(*query: str*) → int

Sends the query to the instrument and returns the response as integer.

query_float(*query: str*) → float

Sends the query to the instrument and returns the response as float.

write_str_with_opc(*cmd: str*, *timeout: int = None*) → None

Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

write_with_opc(*cmd: str*, *timeout: int = None*) → None

This method is an alias to the write_str_with_opc(). Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

query_str_with_opc(*query: str*, *timeout: int = None*) → str

Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_with_opc(*query: str*, *timeout: int = None*) → str

This method is an alias to the query_str_with_opc(). Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_bool_with_opc(*query: str*, *timeout: int = None*) → bool

Sends the opc-synced query to the instrument and returns the response as boolean. If you do not provide timeout, the method uses current opc_timeout.

query_int_with_opc(*query: str*, *timeout: int = None*) → int

Sends the opc-synced query to the instrument and returns the response as integer. If you do not provide timeout, the method uses current opc_timeout.

query_float_with_opc(*query: str*, *timeout: int = None*) → float

Sends the opc-synced query to the instrument and returns the response as float. If you do not provide timeout, the method uses current opc_timeout.

write_bin_block(*cmd: str*, *payload: bytes*) → None

Writes all the payload as binary data block to the instrument. The binary data header is added at the beginning of the transmission automatically, do not include it in the payload!!!

query_bin_block(*query: str*) → bytes

Queries binary data block to bytes. Throws an exception if the returned data was not a binary data. Returns data:bytes

query_bin_block_with_opc(*query: str*, *timeout: int = None*) → bytes

Sends a OPC-synced query and returns binary data block to bytes. If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_float_list(*query: str*) → List[float]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_float_list_with_opc(*query: str*, *timeout: int = None*) → List[float]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_int_list(*query: str*) → List[int]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_int_list_with_opc(*query: str*, *timeout: int = None*) → List[int]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_block_to_file(*query: str*, *file_path: str*, *append: bool = False*) → None

Queries binary data block to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data. Example for transferring a file from Instrument -> PC: query = f'MMEM:DATA? '{INSTR_FILE_PATH}'". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

query_bin_block_to_file_with_opc(*query: str*, *file_path: str*, *append: bool = False*, *timeout: int = None*) → None

Sends a OPC-synced query and writes the returned data to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data.

write_bin_block_from_file(*cmd: str*, *file_path: str*) → None

Writes data from the file as binary data block to the instrument using the provided command. Example for transferring a file from PC -> Instrument: cmd = f'MMEM:DATA '{INSTR_FILE_PATH}',". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

send_file_from_pc_to_instrument(*source_pc_file: str*, *target_instr_file: str*) → None

SCPI Command: MMEM:DATA

Sends file from PC to the instrument

read_file_from_instrument_to_pc(*source_instr_file: str*, *target_pc_file: str*, *append_to_pc_file: bool = False*) → None

SCPI Command: MMEM:DATA?

Reads file from instrument to the PC.

Set the `append_to_pc_file` to True if you want to append the read content to the end of the existing PC file

get_last_sent_cmd() → str

Returns the last commands sent to the instrument. Only works in simulation mode

go_to_local() → None

Puts the instrument into local state.

go_to_remote() → None

Puts the instrument into remote state.

get_lock() → RLock

Returns the thread lock for the current session.

By default:

- If you create standard new RsCMPX_NiotMeas instance with new VISA session, the session gets a new thread lock. You can assign it to other RsCMPX_NiotMeas sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsCMPX_NiotMeas from an existing session, the thread lock is shared automatically making both instances multi-thread safe.

You can always assign new thread lock by calling `driver.utilities.assign_lock()`

assign_lock(*lock: RLock*) → None

Assigns the provided thread lock.

clear_lock()

Clears the existing thread lock, making the current session thread-independent from others that might share the current thread lock.

sync_from(*source: Utilities*) → None

Synchronises these Utils with the source.

CHAPTER

EIGHT

RSCMPX_NIOTMEAS LOGGER

Check the usage in the Getting Started chapter *here*.

class ScpiLogger

Base class for SCPI logging

mode

Sets the logging ON or OFF. Additionally, you can set the logging ON only for errors. Possible values:

- LoggingMode.Off - logging is switched OFF
- LoggingMode.On - logging is switched ON
- LoggingMode.Errors - logging is switched ON, but only for error entries
- LoggingMode.Default - sets the logging to default - the value you have set with logger.default_mode

default_mode

Sets / returns the default logging mode. You can recall the default mode by calling the logger.mode = LoggingMode.Default.

Data Type
LoggingMode

device_name: str

Use this property to change the resource name in the log from the default Resource Name (e.g. TCPIP::192.168.2.101::INSTR) to another name e.g. 'MySigGen1'.

set_logging_target(*target*, *console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target - the target must implement write() and flush(). You can optionally set the console and UDP logging ON or OFF. This method switches the logging target global OFF.

get_logging_target()

Based on the global_mode, it returns the logging target: either the local or the global one.

set_logging_target_global(*console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target to global. The global target must be defined. You can optionally set the console and UDP logging ON or OFF.

log_to_console

Returns logging to console status.

log_to_udp

Returns logging to UDP status.

log_to_console_and_udp

Returns true, if both logging to UDP and console in are True.

info_raw(*log_entry: str*, *add_new_line: bool = True*) → None

Method for logging the raw string without any formatting.

info(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one info entry. For binary log_string, use the info_bin()

error(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one error entry.

set_relative_timestamp(*timestamp: datetime*) → None

If set, the further timestamps will be relative to the entered time.

set_relative_timestamp_now() → None

Sets the relative timestamp to the current time.

get_relative_timestamp() → datetime

Based on the global_mode, it returns the relative timestamp: either the local or the global one.

clear_relative_timestamp() → None

Clears the reference time, and the further logging continues with absolute times.

flush() → None

Flush all the entries.

log_status_check_ok

Sets / returns the current status of status checking OK. If True (default), the log contains logging of the status checking 'Status check: OK'. If False, the 'Status check: OK' is skipped - the log is more compact. Errors will still be logged.

clear_cached_entries() → None

Clears potential cached log entries. Cached log entries are generated when the Logging is ON, but no target has been defined yet.

set_format_string(*value: str*, *line_divider: str = '\n'*) → None

Sets new format string and line divider. If you just want to set the line divider, set the format string value=None The original format string is: `PAD_LEFT12(%START_TIME%) PAD_LEFT25(%DEVICE_NAME%) PAD_LEFT12(%DURATION%) %LOG_STRING_INFO%: %LOG_STRING%`

restore_format_string() → None

Restores the original format string and the line divider to LF

abbreviated_max_len_ascii: int

Defines the maximum length of one ASCII log entry. Default value is 200 characters.

abbreviated_max_len_bin: int

Defines the maximum length of one Binary log entry. Default value is 2048 bytes.

abbreviated_max_len_list: int

Defines the maximum length of one list entry. Default value is 100 elements.

bin_line_block_size: int

Defines number of bytes to display in one line. Default value is 16 bytes.

udp_port

Returns udp logging port.

target_auto_flushing

Returns status of the auto-flushing for the logging target.

CHAPTER

NINE

RSCMPX_NIOTMEAS EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

TEN

INDEX

INDEX

D

E

F

G

I

L

M

R

S

T

U