RsCmwEvdoSig

Release 3.8.10.18

Rohde & Schwarz

May 27, 2021

CONTENTS:


CMW

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

RsCmwEvdoSig 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('3.7.90.32')
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.source_set(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 (in case of big data transfer)

- Multithreading session locking - you can use multiple threads talking to one instrument at the same time

1.2 Installation

RsCmwEvdoSig 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 RsCmwEvdoSig`

Option 2 - Installing in Pycharm

- In Pycharm Menu `File->Settings->Project->Project Interpreter` click on the **'+'** button on the bottom left
- Type `RsCmwEvdoSig` 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 5 easy step for installing the RsCmwEvdoSig offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwEvdoSig needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwEvdoSig package to your computer from the pypi.org: https://pypi.org/project/RsCmwEvdoSig/#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\RsCmwEvdoSig-3.8.10.18.tar`

1.3 Finding Available Instruments

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

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

from RsCmwEvdoSig import *

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

1.4 Initiating Instrument Session

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

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

- Installed RsCmwEvdoSig Python module Version 3.8.10 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCmwEvdoSig import *

# A good practice is to assure that you have a certain minimum version installed
RsCmwEvdoSig.assert_minimum_version('3.8.10')
resource_string_1 = 'TCPIP::192.168.2.101::INSTR'  # Standard LAN connection (also␣
↪called VXI-11)
resource_string_2 = 'TCPIP::192.168.2.101::hislip0'  # Hi-Speed LAN connection - see␣
↪1MA208
resource_string_3 = 'GPIB::20::INSTR'  # GPIB Connection
resource_string_4 = 'USB::0x0AAD::0x0119::022019943::INSTR'  # USB-TMC (Test and␣
↪Measurement Class)

# Initializing the session
driver = RsCmwEvdoSig(resource_string_1)

idn = driver.utilities.query_str('*IDN?')
print(f"\nHello, I am: '{idn}'")
print(f'RsCmwEvdoSig 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 2021.

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

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

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

from RsCmwEvdoSig import *

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

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

from RsCmwEvdoSig import *

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

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

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

Simulating Session

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

```
driver = RsCmwEvdoSig('TCPIP::192.168.56.101::HISLIP', True, True, "Simulate=True")
```

More `option_string` tokens are separated by comma:

```
driver = RsCmwEvdoSig('TCPIP::192.168.56.101::HISLIP', 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 RsCmwEvdoSig objects:

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

from RsCmwEvdoSig import *

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

1.5 Plain SCPI Communication

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

driver = RsCmwEvdoSig('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 RsCmwEvdoSig import *

driver = RsCmwEvdoSig('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 RsCmwEvdoSig raises an exception. Speaking of exceptions, an important feature of the RsCmwEvdoSig 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 RsCmwEvdoSig import *

driver = RsCmwEvdoSig('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
```

1.6 Error Checking

RsCmwEvdoSig 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.

1.7 Exception Handling

The base class for all the exceptions raised by the RsCmwEvdoSig 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 RsCmwEvdoSig import *

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

(continues on next page)

(continued from previous page)

```
    driver = RsCmwEvdoSig('TCPIP::10.112.1.179::HISLIP')
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 RsCmwEvdoSig exceptions
    print(e.args[0])
    print('Some other RsCmwEvdoSig 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.

1.8 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 RsCmwEvdoSig, 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 RsCmwEvdoSig 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')
```

1.9 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")
```

1.10 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 RsCmwEvdoSig has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

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

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

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

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

1.11 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, RsCmwEvdoSig 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 RsCmwEvdoSig object
"""

import threading
from RsCmwEvdoSig import *


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


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

(continues on next page)

(continued from previous page)

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsCmwEvdoSig import *


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

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

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

(continues on next page)

(continued from previous page)

```
print('All threads ended')

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

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

Multiple instrument sessions accessed from multiple threads

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

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

Run the following example:

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

import threading
from RsCmwEvdoSig import *


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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

CHAPTER

TWO

REVISION HISTORY

Rohde & Schwarz CMW Base System RsCmwBase instrument driver.

Supported instruments: CMW500, CMW100, CMW270, CMW280

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

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

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

Currently supported CMW subsystems:

- Base: RsCmwBase
- Global Purpose RF: RsCmwGprfGen, RsCmwGprfMeas
- Bluetooth: RsCmwBluetoothSig, RsCmwBluetoothMeas
- LTE: RsCmwLteSig, RsCmwLteMeas
- CDMA2000: RsCdma2kSig, RsCdma2kMeas
- 1xEVDO: RsCmwEvdoSig, RsCmwEvdoMeas
- WCDMA: RsCmwWcdmaSig, RsCmwWcdmaMeas
- GSM: RsCmwGsmSig, RsCmwGsmMeas
- WLAN: RsCmwWlanSig, RscmwWlanMeas
- DAU: RsCMwDau

In case you require support for more subsystems, please contact our customer support on customersupport@rohde-schwarz.com with the topic "Auto-generated Python drivers" in the email subject. This will speed up the response process

---

Examples: Download the file 'CMW Python instrument drivers' from https://www.rohde-schwarz.com/driver/cmw500_overview/ The zip file contains the examples on how to use these drivers. Remember to adjust the resourceName string to fit your instrument.

Release Notes for the whole RsCmwXXX group:

Latest release notes summary: <INVALID>

Version 3.7.90.39

- <INVALID>

Version 3.8.xx2

- Fixed several misspelled arguments and command headers

Version 3.8.xx1

- Bluetooth and WLAN update for FW versions 3.8.xxx

Version 3.7.xx8

- Added documentation on ReadTheDocs

Version 3.7.xx7

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

Version 3.7.xx6

- Added new UDF integer number recognition

Version 3.7.xx5

- Added RsCmwDau

Version 3.7.xx4

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

Version 3.7.xx3

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

Version 3.7.xx2

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

Version 1.0.0.0

- First released version

CHAPTER

THREE

ENUMS

3.1 AccessDuration

```
# Example value:
value = enums.AccessDuration.S128
# All values (4x):
S128 | S16 | S32 | S64
```

3.2 ApplicationMode

```
# Example value:
value = enums.ApplicationMode.FAR
# All values (4x):
FAR | FWD | PACKet | REV
```

3.3 ApplyTimeAt

```
# Example value:
value = enums.ApplyTimeAt.EVER
# All values (3x):
EVER | NEXT | SUSO
```

3.4 AutoManualMode

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

3.5 AwgnMode

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

3.6 BandClass

```
# First value:
value = enums.BandClass.AWS
# Last value:
value = enums.BandClass.USPC
# All values (23x):
AWS | B18M | IEXT | IM2K | JTAC | KCEL | KPCS | LBANd
LO7C | N45T | NA7C | NA8S | NA9C | NAPC | PA4M | PA8M
PS7C | SBANd | TACS | U25B | U25F | USC | USPC
```

3.7 CarrierStatus

```
# Example value:
value = enums.CarrierStatus.INACtive
# All values (4x):
INACtive | OK | STALe | VIOLated
```

3.8 ConnectionState

```
# Example value:
value = enums.ConnectionState.CONNected
# All values (7x):
CONNected | IDLE | OFF | ON | PAGing | SNEGotiation | SOPen
```

3.9 CswitchedAction

```
# Example value:
value = enums.CswitchedAction.CLOSe
# All values (4x):
CLOSe | CONNect | DISConnect | HANDoff
```

3.10 CtrlChannelDataRate

```
# Example value:
value = enums.CtrlChannelDataRate.R384
# All values (2x):
R384 | R768
```

3.11 DisplayTab

```
# Example value:
value = enums.DisplayTab.CTRLchper
# All values (6x):
CTRLchper | DATA | OVERview | PER | RLQ | THRoughput
```

3.12 ExpPowerMode

```
# Example value:
value = enums.ExpPowerMode.AUTO
# All values (5x):
AUTO | MANual | MAX | MIN | OLRule
```

3.13 Fmode

```
# Example value:
value = enums.Fmode.AALWays
# All values (3x):
AALWays | NAALways | NUSed
```

3.14 FsimStandard

```
# Example value:
value = enums.FsimStandard.P1
# All values (5x):
P1 | P2 | P3 | P4 | P5
```

3.15 InsertLossMode

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

3.16 IpAddressIndex

```
# Example value:
value = enums.IpAddressIndex.IP1
# All values (3x):
IP1 | IP2 | IP3
```

3.17 KeepConstant

```
# Example value:
value = enums.KeepConstant.DSHift
# All values (2x):
DSHift | SPEed
```

3.18 LinkCarrier

```
# Example value:
value = enums.LinkCarrier.ACTive
# All values (4x):
ACTive | DISabled | NACTive | NCConnected
```

3.19 LogCategory

```
# Example value:
value = enums.LogCategory.CONTinue
# All values (4x):
CONTinue | ERRor | INFO | WARNing
```

3.20 LteBand

```
# First value:
value = enums.LteBand.OB1
# Last value:
value = enums.LteBand.UDEFined
# All values (45x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16
OB17 | OB18 | OB19 | OB2 | OB20 | OB21 | OB22 | OB23
OB24 | OB25 | OB26 | OB27 | OB28 | OB29 | OB3 | OB30
OB31 | OB32 | OB33 | OB34 | OB35 | OB36 | OB37 | OB38
OB39 | OB4 | OB40 | OB41 | OB42 | OB43 | OB44 | OB5
OB6 | OB7 | OB8 | OB9 | UDEFined
```

3.21 MainGenState

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

3.22 NetworkRelease

```
# Example value:
value = enums.NetworkRelease.R0
# All values (3x):
R0 | RA | RB
```

3.23 NetworkSegment

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

3.24 PacketSize

```
# First value:
value = enums.PacketSize.S128
# Last value:
value = enums.PacketSize.TOTal
# All values (12x):
```

(continues on next page)

(continued from previous page)

```
S128 | S1K | S256 | S2K | S3K | S4K | S512 | S5K
S6K | S7K | S8K | TOTal
```

3.25 PdState

```
# Example value:
value = enums.PdState.CONNected
# All values (4x):
CONNected | DORMant | OFF | ON
```

3.26 PerEvaluation

```
# Example value:
value = enums.PerEvaluation.ALLCarriers
# All values (2x):
ALLCarriers | PERCarrier
```

3.27 PerStopCondition

```
# Example value:
value = enums.PerStopCondition.ALEXceeded
# All values (4x):
ALEXceeded | MCLexceeded | MPERexceeded | NONE
```

3.28 PlSlots

```
# Example value:
value = enums.PlSlots.S16
# All values (2x):
S16 | S4
```

3.29 PlSubtype

```
# Example value:
value = enums.PlSubtype.ST01
# All values (3x):
ST01 | ST2 | ST3
```

3.30 PowerCtrlBits

```
# Example value:
value = enums.PowerCtrlBits.ADOWn
# All values (6x):
ADOWn | AUP | AUTO | HOLD | PATTern | RTESt
```

3.31 PrefApplication

```
# Example value:
value = enums.PrefApplication.DPA
# All values (2x):
DPA | EMPA
```

3.32 PrefAppMode

```
# Example value:
value = enums.PrefAppMode.EHRPd
# All values (2x):
EHRPd | HRPD
```

3.33 ProbesAckMode

```
# Example value:
value = enums.ProbesAckMode.ACKN
# All values (2x):
ACKN | IGN
```

3.34 Repeat

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

3.35 ResourceState

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

3.36 RevLinkPerDataRate

```
# First value:
value = enums.RevLinkPerDataRate.R0K0
# Last value:
value = enums.RevLinkPerDataRate.TOTal
# All values (14x):
R0K0 | R115k2 | R1228k8 | R153k6 | R1843k2 | R19K2 | R230k4 | R307k2
R38K4 | R460k8 | R614k4 | R76K8 | R921k6 | TOTal
```

3.37 RxConnector

```
# First value:
value = enums.RxConnector.I11I
# Last value:
value = enums.RxConnector.RH8
# All values (154x):
I11I | I13I | I15I | I17I | I21I | I23I | I25I | I27I
I31I | I33I | I35I | I37I | I41I | I43I | I45I | I47I
IF1 | IF2 | IF3 | IQ1I | IQ3I | IQ5I | IQ7I | R11
R11C | R12 | R12C | R12I | R13 | R13C | R14 | R14C
R14I | R15 | R16 | R17 | R18 | R21 | R21C | R22
R22C | R22I | R23 | R23C | R24 | R24C | R24I | R25
R26 | R27 | R28 | R31 | R31C | R32 | R32C | R32I
R33 | R33C | R34 | R34C | R34I | R35 | R36 | R37
R38 | R41 | R41C | R42 | R42C | R42I | R43 | R43C
R44 | R44C | R44I | R45 | R46 | R47 | R48 | RA1
RA2 | RA3 | RA4 | RA5 | RA6 | RA7 | RA8 | RB1
RB2 | RB3 | RB4 | RB5 | RB6 | RB7 | RB8 | RC1
RC2 | RC3 | RC4 | RC5 | RC6 | RC7 | RC8 | RD1
RD2 | RD3 | RD4 | RD5 | RD6 | RD7 | RD8 | RE1
RE2 | RE3 | RE4 | RE5 | RE6 | RE7 | RE8 | RF1
RF1C | RF2 | RF2C | RF2I | RF3 | RF3C | RF4 | RF4C
RF4I | RF5 | RF5C | RF6 | RF6C | RF7 | RF8 | RFAC
RFBC | RFBI | RG1 | RG2 | RG3 | RG4 | RG5 | RG6
RG7 | RG8 | RH1 | RH2 | RH3 | RH4 | RH5 | RH6
RH7 | RH8
```

3.38 RxConverter

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

3.39 RxSignalState

```
# Example value:
value = enums.RxSignalState.HIGH
# All values (4x):
HIGH | LOW | NAV | OK
```

3.40 SampleRate

```
# Example value:
value = enums.SampleRate.M1
# All values (8x):
M1 | M100 | M15 | M19 | M3 | M30 | M7 | M9
```

3.41 SamRate

```
# Example value:
value = enums.SamRate.R19K
# All values (3x):
R19K | R38K | R9K
```

3.42 Scenario

```
# Example value:
value = enums.Scenario.HMFading
# All values (6x):
HMFading | HMLite | HMODe | SCELl | SCFading | UNDefined
```

3.43 SectorIdFormat

```
# Example value:
value = enums.SectorIdFormat.A41N
# All values (2x):
A41N | MANual
```

3.44 SegmentBits

```
# Example value:
value = enums.SegmentBits.ALTernating
# All values (3x):
ALTernating | DOWN | UP
```

3.45 SlopeType

```
# Example value:
value = enums.SlopeType.NEGative
# All values (2x):
NEGative | POSitive
```

3.46 SourceInt

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

3.47 SyncState

```
# Example value:
value = enums.SyncState.ADINtermed
# All values (7x):
ADINtermed | ADJusted | INValid | OFF | ON | PENDing | RFHandover
```

3.48 T2Pmode

```
# Example value:
value = enums.T2Pmode.RFCO
# All values (2x):
RFCO | TPUT
```

3.49 TimeSource

```
# Example value:
value = enums.TimeSource.CMWTime
# All values (3x):
CMWTime | DATE | SYNC
```

3.50 TxConnector

```
# First value:
value = enums.TxConnector.I12O
# Last value:
value = enums.TxConnector.RH18
# All values (77x):
I12O | I14O | I16O | I18O | I22O | I24O | I26O | I28O
I32O | I34O | I36O | I38O | I42O | I44O | I46O | I48O
IF1 | IF2 | IF3 | IQ2O | IQ4O | IQ6O | IQ8O | R118
R1183 | R1184 | R11C | R11O | R11O3 | R11O4 | R12C | R13C
R13O | R14C | R214 | R218 | R21C | R21O | R22C | R23C
R23O | R24C | R258 | R318 | R31C | R31O | R32C | R33C
R33O | R34C | R418 | R41C | R41O | R42C | R43C | R43O
R44C | RA18 | RB14 | RB18 | RC18 | RD18 | RE18 | RF18
RF1C | RF1O | RF2C | RF3C | RF3O | RF4C | RF5C | RF6C
RFAC | RFAO | RFBC | RG18 | RH18
```

3.51 TxConverter

```
# First value:
value = enums.TxConverter.ITX1
# Last value:
value = enums.TxConverter.TX44
# All values (40x):
ITX1 | ITX11 | ITX12 | ITX13 | ITX14 | ITX2 | ITX21 | ITX22
ITX23 | ITX24 | ITX3 | ITX31 | ITX32 | ITX33 | ITX34 | ITX4
ITX41 | ITX42 | ITX43 | ITX44 | TX1 | TX11 | TX12 | TX13
TX14 | TX2 | TX21 | TX22 | TX23 | TX24 | TX3 | TX31
TX32 | TX33 | TX34 | TX4 | TX41 | TX42 | TX43 | TX44
```

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 CellNo

```
# First value:
value = repcap.CellNo.Nr1
# Range:
Nr1 .. Nr16
# All values (16x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
```

4.3 IpAddress

```
# First value:
value = repcap.IpAddress.Version4
# Values (2x):
Version4 | Version6
```

4.4 NeighborCell

```
# First value:
value = repcap.NeighborCell.Nr1
# Range:
Nr1 .. Nr16
# All values (16x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
```

4.5 Path

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

4.6 Segment

```
# First value:
value = repcap.Segment.S1
# Values (4x):
S1 | S2 | S3 | S4
```

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('3.7.90.32')
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:
↪{event_args.message}')
```

(continues on next page)

(continued from previous page)

```
# 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.source_set(enums.SourceIntExt.INTernal)

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

CHAPTER

SIX

INDEX

CHAPTER

SEVEN

RSCMWEVDOSIG API STRUCTURE

Global RepCaps

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

class RsCmwEvdoSig(*resource_name: str*, *id_query: bool = True*, *reset: bool = False*, *options: Optional[str] = None*, *direct_session: Optional[object] = None*)

285 total commands, 10 Sub-groups, 0 group commands

Initializes new RsCmwEvdoSig 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=ni' - forces usage of National Instruments Visa
- 'QueryInstrumentStatus = False' - same as driver.utilities.instrument_status_checking = False
- 'DriverSetup=(WriteDelay = 20, ReadDelay = 5)' - Introduces delay of 20ms before each write and 5ms before each read
- 'DriverSetup=(OpcWaitMode = OpcQuery)' - mode for all the opc-synchronised write/reads. Other modes: StbPolling, StbPollingSlow, StbPollingSuperSlow
- 'DriverSetup=(AddTermCharToWriteBinBLock = True)' - Adds one additional LF to the end of the binary data (some instruments require that)
- 'DriverSetup=(AssureWriteWithTermChar = True)' - Makes sure each command/query is terminated with termination character. Default: Interface dependent
- 'DriverSetup=(TerminationCharacter = 'x')' - Sets the termination character for reading. Default: '<LF>' (LineFeed)
- 'DriverSetup=(IoSegmentSize = 10E3)' - Maximum size of one write/read segment. If transferred data is bigger, it is split to more segments
- 'DriverSetup=(OpcTimeout = 10000)' - same as driver.utilities.opc_timeout = 10000
- 'DriverSetup=(VisaTimeout = 5000)' - same as driver.utilities.visa_timeout = 5000

- 'DriverSetup=(ViClearExeMode = 255)' - Binary combination where 1 means performing viClear() on a certain interface as the very first command in init
- 'DriverSetup=(OpcQueryAfterWrite = True)' - same as driver.utilities.opc_query_after_write = True

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.

close() → None

Closes the active RsCmwEvdoSig session.

classmethod from_existing_session(*session: object*, *options: Optional[str] = None*) → RsCmwEvdoSig

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

Parameters

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

get_session_handle() → object

Returns the underlying session handle.

static list_resources(*expression: str = '?*::INSTR'*, *visa_select: Optional[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: '@ni', '@rs'

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

Subgroups

7.1 Configure

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:DISPlay
CONFigure:EVDO:SIGNaling<Instance>:ETOE
```

class Configure

Configure commands group definition. 220 total commands, 20 Sub-groups, 2 group commands

get_display() → RsCmwEvdoSig.enums.DisplayTab

```
# SCPI: CONFigure:EVDO:SIGNaling<Instance>:DISPlay
value: enums.DisplayTab = driver.configure.get_display()
```

Selects the view to be shown when the display is switched on during remote control.

return tab: PER | THRoughput | DATA | OVERview 'RX Meas': 'PER', 'Throughput', 'Data'; 1xEV-DO signaling: overview

get_etoe() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:ETOE
value: bool = driver.configure.get_etoe()
```

Enables the setup of a connection between the signaling unit and the data application unit (DAU) , required for IP-based data tests involving the DAU.

return end_to_end_enable: OFF | ON

set_display(*tab: RsCmwEvdoSig.enums.DisplayTab*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<Instance>:DISPlay
driver.configure.set_display(tab = enums.DisplayTab.CTRLchper)
```

Selects the view to be shown when the display is switched on during remote control.

param tab PER | THRoughput | DATA | OVERview 'RX Meas': 'PER', 'Throughput', 'Data'; 1xEV-DO signaling: overview

set_etoe(*end_to_end_enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:ETOE
driver.configure.set_etoe(end_to_end_enable = False)
```

Enables the setup of a connection between the signaling unit and the data application unit (DAU) , required for IP-based data tests involving the DAU.

param end_to_end_enable OFF | ON

Cloning the Group

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

Subgroups

7.1.1 Test

class Test

Test commands group definition. 2 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.1.1 Cstatus

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:TEST:CSTatus:MEID
CONFigure:EVDO:SIGNaling<Instance>:TEST:CSTatus:ESN
```

class Cstatus

Cstatus commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_esn() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:TEST:CSTatus:ESN
value: float = driver.configure.test.cstatus.get_esn()
```

No command help available

return esn: No help available

get_meid() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:TEST:CSTatus:MEID
value: float = driver.configure.test.cstatus.get_meid()
```

No command help available

return meid: No help available

set_esn(*esn: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:TEST:CSTatus:ESN
driver.configure.test.cstatus.set_esn(esn = 1.0)
```

No command help available

param esn No help available

set_meid(*meid: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:TEST:CSTatus:MEID
driver.configure.test.cstatus.set_meid(meid = 1.0)
```

No command help available

param meid No help available

7.1.2 RfSettings

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RFSettings:EATTenuation
CONFigure:EVDO:SIGNaling<Instance>:RFSettings:BCLass
CONFigure:EVDO:SIGNaling<Instance>:RFSettings:FREQuency
CONFigure:EVDO:SIGNaling<Instance>:RFSettings:FLFRequency
CONFigure:EVDO:SIGNaling<Instance>:RFSettings:RLFRequency
CONFigure:EVDO:SIGNaling<Instance>:RFSettings:FOFFset
CONFigure:EVDO:SIGNaling<Instance>:RFSettings:CHANnel
```

class RfSettings

RfSettings commands group definition. 7 total commands, 0 Sub-groups, 7 group commands

class EattenuationStruct

Structure for reading output parameters. Fields:

- Rf_Input_Ext_Att: float: TX external attenuation Range: -50 dB to 90 dB
- Rf_Output_Ext_Att: float: RX external attenuation Range: -50 dB to 90 dB, Unit: dB

class FrequencyStruct

Structure for reading output parameters. Fields:

- Forward_Link_Freq: float: Range: 0 Hz to 6.1 GHz , Unit: Hz
- Reverse_Link_Freq: float: Range: 0 Hz to 6.1 GHz , Unit: Hz

get_bclass() → RsCmwEvdoSig.enums.BandClass

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:BCLass
value: enums.BandClass = driver.configure.rfSettings.get_bclass()
```

Selects the band class for the cell simulated by the signaling application. See also: 'Band Classes'

return band_class: USC | KCEL | NAPC | TACS | JTAC | KPCS | N45T | IM2K | NA7C | B18M | NA9C | NA8S | PA4M | PA8M | IEXT | USPC | AWS | U25B | U25F | PS7C | LO7C | LBANd | SBANd USC: BC 0, US-Cellular KCEL: BC 0, Korean Cellular NAPC: BC 1, North American PCS TACS: BC 2, TACS Band JTAC: BC 3, JTACS

Band KPCS: BC 4, Korean PCS N45T: BC 5, NMT-450 IM2K: BC 6, IMT-2000 NA7C: BC 7, Upper 700 MHz B18M: BC 8, 1800 MHz Band NA9C: BC 9, North American 900 MHz NA8S: BC 10, Secondary 800 MHz PA4M: BC 11, European 400 MHz PAMR PA8M: BC 12, 800 MHz PAMR IEXT: BC 13, IMT-2000 2.5 GHz Extension USPC: BC 14, US PCS 1900 MHz AWS: BC 15, AWS Band U25B: BC 16, US 2.5 GHz Band U25F: BC 17, US 2.5 GHz Forward PS7C: BC 18, Public Safety Band 700 MHz LO7C: BC 19, Lower 700 MHz LBAN: BC 20, L-Band SBAN: BC 21, S-Band

get_channel() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:CHANnel
value: int = driver.configure.rfSettings.get_channel()
```

Sets/gets the main RF channel (the only one for network releases 0/A) for 1xEV-DO signaling tests. The reset value and the range of possible values depend on the selected band class. The values below are for band class BC0 (US-Cellular) . For an overview, see 'Band Classes'.

return channel: Range: 1 to 799, 991 to 1323 , Unit: Channel no.

get_eattenuation() → EattenuationStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:EATTenuation
value: EattenuationStruct = driver.configure.rfSettings.get_eattenuation()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

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

get_fl_frequency() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:FLFRequency
value: float = driver.configure.rfSettings.get_fl_frequency()
```

Queries the forward link frequency, depending on the selected band class and channel (method RsCmwEvdoSig.Configure. RfSettings.bclass, method RsCmwEvdoSig.Configure.RfSettings.channel) .

return frequency: Range: 0 Hz to 6.1 GHz, Unit: Hz

get_foffset() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:FOFFset
value: float = driver.configure.rfSettings.get_foffset()
```

Modifies the nominal forward link frequency of the selected band class and RF channel by a frequency offset.

return freq_offset: Range: -50 kHz to 50 kHz, Unit: Hz

get_frequency() → FrequencyStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:FREQuency
value: FrequencyStruct = driver.configure.rfSettings.get_frequency()
```

Queries the forward and reverse link frequency, depending on the selected band class and channel.

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

get_rl_frequency() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:RLFRequency
value: float = driver.configure.rfSettings.get_rl_frequency()
```

Queries the reverse link frequency, depending on the selected band class and channel (method RsCmwEvdoSig.Configure. RfSettings.bclass, method RsCmwEvdoSig.Configure.RfSettings.channel) .

return frequency: Range: 0 Hz to 6.1 GHz, Unit: Hz

set_bclass(*band_class: RsCmwEvdoSig.enums.BandClass*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:BCLass
driver.configure.rfSettings.set_bclass(band_class = enums.BandClass.AWS)
```

Selects the band class for the cell simulated by the signaling application. See also: 'Band Classes'

param band_class USC | KCEL | NAPC | TACS | JTAC | KPCS | N45T | IM2K | NA7C | B18M | NA9C | NA8S | PA4M | PA8M | IEXT | USPC | AWS | U25B | U25F | PS7C | LO7C | LBANd | SBANd USC: BC 0, US-Cellular KCEL: BC 0, Korean Cellular NAPC: BC 1, North American PCS TACS: BC 2, TACS Band JTAC: BC 3, JTACS Band KPCS: BC 4, Korean PCS N45T: BC 5, NMT-450 IM2K: BC 6, IMT-2000 NA7C: BC 7, Upper 700 MHz B18M: BC 8, 1800 MHz Band NA9C: BC 9, North American 900 MHz NA8S: BC 10, Secondary 800 MHz PA4M: BC 11, European 400 MHz PAMR PA8M: BC 12, 800 MHz PAMR IEXT: BC 13, IMT-2000 2.5 GHz Extension USPC: BC 14, US PCS 1900 MHz AWS: BC 15, AWS Band U25B: BC 16, US 2.5 GHz Band U25F: BC 17, US 2.5 GHz Forward PS7C: BC 18, Public Safety Band 700 MHz LO7C: BC 19, Lower 700 MHz LBAN: BC 20, L-Band SBAN: BC 21, S-Band

set_channel(*channel: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:CHANnel
driver.configure.rfSettings.set_channel(channel = 1)
```

Sets/gets the main RF channel (the only one for network releases 0/A) for 1xEV-DO signaling tests. The reset value and the range of possible values depend on the selected band class. The values below are for band class BC0 (US-Cellular) . For an overview, see 'Band Classes'.

param channel Range: 1 to 799, 991 to 1323 , Unit: Channel no.

set_eattenuation(*value: RsCmwEvdoSig.Implementations.Configure_.RfSettings.RfSettings.EattenuationStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:EATTenuation
driver.configure.rfSettings.set_eattenuation(value = EattenuationStruct())
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

param value see the help for EattenuationStruct structure arguments.

set_foffset(*freq_offset: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFSettings:FOFFset
driver.configure.rfSettings.set_foffset(freq_offset = 1.0)
```

Modifies the nominal forward link frequency of the selected band class and RF channel by a frequency offset.

param freq_offset Range: -50 kHz to 50 kHz, Unit: Hz

7.1.3 Fading

class Fading

Fading commands group definition. 17 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.3.1 Fsimulator

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:KCONstant
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:ENABle
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:STANdard
```

class Fsimulator

Fsimulator commands group definition. 9 total commands, 3 Sub-groups, 3 group commands

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:ENABle
value: bool = driver.configure.fading.fsimulator.get_enable()
```

Enables/disables the fading simulator.

return enable: OFF | ON

get_kconstant() → RsCmwEvdoSig.enums.KeepConstant

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:KCONstant
value: enums.KeepConstant = driver.configure.fading.fsimulator.get_kconstant()
```

No command help available

return keep_constant: No help available

get_standard() → RsCmwEvdoSig.enums.FsimStandard

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:STANdard
value: enums.FsimStandard = driver.configure.fading.fsimulator.get_standard()
```

Selects one of the propagation conditions defined in the table 6.4.1-1 of 3GPP2 C.S0032.

return standard: P1 | P2 | P3 | P4 | P5 EVDO1 to EVDO5 P1: Two paths, speed 15 km/h (band classes 5, 11) , 8 km/h (other band classes) P2: One path, speed 3 km/h, exception: 6 km/h for band classes 5, 11 P3: One path, speed 30 km/h, exception: 58 km/h for band classes 5, 11 P4: Three paths, speed 100 km/h, exception: 192 km/h for band classes 5, 11 P5: Two paths, speed 0 km/h

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:ENABle
driver.configure.fading.fsimulator.set_enable(enable = False)
```

Enables/disables the fading simulator.

param enable OFF | ON

set_kconstant(*keep_constant: RsCmwEvdoSig.enums.KeepConstant*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:KCONstant
driver.configure.fading.fsimulator.set_kconstant(keep_constant = enums.
↪KeepConstant.DSHift)
```

No command help available

param keep_constant No help available

set_standard(*standard: RsCmwEvdoSig.enums.FsimStandard*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:STANdard
driver.configure.fading.fsimulator.set_standard(standard = enums.FsimStandard.
↪P1)
```

Selects one of the propagation conditions defined in the table 6.4.1-1 of 3GPP2 C.S0032.

param standard P1 | P2 | P3 | P4 | P5 EVDO1 to EVDO5 P1: Two paths, speed 15 km/h (band classes 5, 11) , 8 km/h (other band classes) P2: One path, speed 3 km/h, exception: 6 km/h for band classes 5, 11 P3: One path, speed 30 km/h, exception: 58 km/h for band classes 5, 11 P4: Three paths, speed 100 km/h, exception: 192 km/h for band classes 5, 11 P5: Two paths, speed 0 km/h

Cloning the Group

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

Subgroups

7.1.3.1.1 Globale

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:GLOBal:SEED
```

class Globale

Globale commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_seed() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:GLOBal:SEED
value: int = driver.configure.fading.fsimulator.globale.get_seed()
```

Sets the start seed for the pseudo-random fading algorithm.

return seed: Range: 0 to 9

set_seed(*seed: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:GLOBal:SEED
driver.configure.fading.fsimulator.globale.set_seed(seed = 1)
```

Sets the start seed for the pseudo-random fading algorithm.

param seed Range: 0 to 9

7.1.3.1.2 Restart

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:RESTart:MODE
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:RESTart
```

class Restart

Restart commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_mode() → RsCmwEvdoSig.enums.AutoManualMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:RESTart:MODE
value: enums.AutoManualMode = driver.configure.fading.fsimulator.restart.get_
↪mode()
```

Sets the restart mode of the fading simulator.

return restart_mode: AUTO | MANual AUTO: fading automatically starts with the DL signal MANual: fading is started and restarted manually (see method RsCmwEvdoSig.Configure.Fading.Fsimulator.Restart.set)

set() → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:RESTart
driver.configure.fading.fsimulator.restart.set()
```

Restarts the fading process in MANual mode (see method RsCmwEvdoSig.Configure.Fading.Fsimulator.Restart.mode) .

set_mode(*restart_mode: RsCmwEvdoSig.enums.AutoManualMode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:RESTart:MODE
driver.configure.fading.fsimulator.restart.set_mode(restart_mode = enums.
↪AutoManualMode.AUTO)
```

Sets the restart mode of the fading simulator.

param restart_mode AUTO | MANual AUTO: fading automatically starts with the DL signal MANual: fading is started and restarted manually (see method RsCmwEvdoSig.Configure.Fading.Fsimulator.Restart.set)

set_with_opc() → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:RESTart
driver.configure.fading.fsimulator.restart.set_with_opc()
```

Restarts the fading process in MANual mode (see method RsCmwEvdoSig.Configure.Fading.Fsimulator.Restart.mode) .

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

7.1.3.1.3 Iloss

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:ILOSs:MODE
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:ILOSs:LOSS
CONFigure:EVDO:SIGNaling<Instance>:FADing:FSIMulator:ILOSs:CSAMples
```

class Iloss

Iloss commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_csamples() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:ILOSs:CSAMples
value: float = driver.configure.fading.fsimulator.iloss.get_csamples()
```

No command help available

return clipped_samples: No help available

get_loss() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:ILOSs:LOSS
value: float = driver.configure.fading.fsimulator.iloss.get_loss()
```

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see method RsCmwEvdoSig. Configure.Fading.Fsimulator.Iloss.mode) .

return insertion_loss: Range: 0 dB to 18 dB , Unit: dB

get_mode() → RsCmwEvdoSig.enums.InsertLossMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:ILOSs:MODE
value: enums.InsertLossMode = driver.configure.fading.fsimulator.iloss.get_
↪mode()
```

Sets the insertion loss mode.

return insert_loss_mode: NORMal | USER NORMal: the insertion loss is determined by the fading profile USER: the insertion loss can be adjusted manually

set_loss(*insertion_loss: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:ILOSs:LOSS
driver.configure.fading.fsimulator.iloss.set_loss(insertion_loss = 1.0)
```

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see method RsCmwEvdoSig. Configure.Fading.Fsimulator.Iloss.mode) .

param insertion_loss Range: 0 dB to 18 dB , Unit: dB

set_mode(*insert_loss_mode: RsCmwEvdoSig.enums.InsertLossMode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:FSIMulator:ILOSs:MODE
driver.configure.fading.fsimulator.iloss.set_mode(insert_loss_mode = enums.
↪InsertLossMode.NORMal)
```

Sets the insertion loss mode.

param insert_loss_mode NORMal | USER NORMal: the insertion loss is determined by the fading profile USER: the insertion loss can be adjusted manually

7.1.3.2 Awgn

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:AWGN:ENABle
CONFigure:EVDO:SIGNaling<Instance>:FADing:AWGN:SNRatio
```

class Awgn

Awgn commands group definition. 4 total commands, 1 Sub-groups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:AWGN:ENABle
value: bool = driver.configure.fading.awgn.get_enable()
```

Enables or disables AWGN insertion via the fading module. For multi-carrier, the same settings are applied to all carriers. Thus it is sufficient to configure one carrier.

return enable: OFF | ON

get_sn_ratio() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:AWGN:SNRatio
value: float = driver.configure.fading.awgn.get_sn_ratio()
```

Queries the signal to noise ratio for the AWGN inserted via the internal fading module.

return ratio: Range: -50 dB to 30 dB , Unit: dB

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:AWGN:ENABle
driver.configure.fading.awgn.set_enable(enable = False)
```

Enables or disables AWGN insertion via the fading module. For multi-carrier, the same settings are applied to all carriers. Thus it is sufficient to configure one carrier.

param enable OFF | ON

set_sn_ratio(*ratio: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:AWGN:SNRatio
driver.configure.fading.awgn.set_sn_ratio(ratio = 1.0)
```

Queries the signal to noise ratio for the AWGN inserted via the internal fading module.

param ratio Range: -50 dB to 30 dB , Unit: dB

Cloning the Group

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

Subgroups

7.1.3.2.1 Bandwidth

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:AWGN:BWIDth:RATio
CONFigure:EVDO:SIGNaling<Instance>:FADing:AWGN:BWIDth:NOISe
```

class Bandwidth

Bandwidth commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_noise() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:AWGN:BWIDth:NOISe
value: float = driver.configure.fading.awgn.bandwidth.get_noise()
```

Queries the bandwidth of the AWGN inserted via the internal fading module.

return noise_bandwidth: Range: 0 Hz to 80 MHz , Unit: Hz

get_ratio() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:AWGN:BWIDth:RATio
value: float = driver.configure.fading.awgn.bandwidth.get_ratio()
```

Queries the AWGN minimal noise to system bandwidth ratio for the AWGN inserted via the internal fading module.

return ratio: Range: 1 to 250

set_ratio(*ratio: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:AWGN:BWIDth:RATio
driver.configure.fading.awgn.bandwidth.set_ratio(ratio = 1.0)
```

Queries the AWGN minimal noise to system bandwidth ratio for the AWGN inserted via the internal fading module.

param ratio Range: 1 to 250

7.1.3.3 Power

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:POWer:SIGNal
CONFigure:EVDO:SIGNaling<Instance>:FADing:POWer:SUM
```

class Power

Power commands group definition. 4 total commands, 1 Sub-groups, 2 group commands

get_signal() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:POWer:SIGNal
value: float = driver.configure.fading.power.get_signal()
```

No command help available

return signal_power: No help available

get_sum() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:POWer:SUM
value: float = driver.configure.fading.power.get_sum()
```

Queries the calculated total power (signal + noise) on the forward link.

return power: Unit: dBm

Cloning the Group

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

Subgroups

7.1.3.3.1 Noise

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:FADing:POWer:NOISe:TOTal
CONFigure:EVDO:SIGNaling<Instance>:FADing:POWer:NOISe
```

class Noise

Noise commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_total() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:POWer:NOISe:TOTal
value: float = driver.configure.fading.power.noise.get_total()
```

Queries the total noise power.

return noise_power: Unit: dBm

get_value() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:FADing:POWer:NOISe
value: float = driver.configure.fading.power.noise.get_value()
```

Queries the calculated system bandwidth noise power on the forward link.

return noise_power: Range: -500 dBm to 500 dBm , Unit: dBm

7.1.4 IqIn

class IqIn

IqIn commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.4.1 Path<Path>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.iqIn.path.repcap_path_get()
driver.configure.iqIn.path.repcap_path_set(repcap.Path.Nr1)
```

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:IQIN:PATH<Path>
```

class Path

Path commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: Path, default value after init: Path.Nr1

class PathStruct

Structure for setting input parameters. Fields:

- Pep: float: Peak envelope power of the incoming baseband signal Range: -60 dBFS to 0 dBFS, Unit: dBFS
- Level: float: Average level of the incoming baseband signal (without noise) Range: depends on crest factor and level of outgoing baseband signal , Unit: dBFS

get(*path=<Path.Default: -1>*) → PathStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:IQIN:PATH<n>
value: PathStruct = driver.configure.iqIn.path.get(path = repcap.Path.Default)
```

Specifies properties of the baseband signal at the I/Q input.

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

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

set(*structure: RsCmwEvdoSig.Implementations.Configure_.IqIn_.Path.Path.PathStruct*, *path=<Path.Default: -1>*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:IQIN:PATH<n>
driver.configure.iqIn.path.set(value = [PROPERTY_STRUCT_NAME](), path = repcap.
↪Path.Default)
```

Specifies properties of the baseband signal at the I/Q input.

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

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

Cloning the Group

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

7.1.5 Carrier

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:CARRier:SETTing
CONFigure:EVDO:SIGNaling<Instance>:CARRier:CHANnel
CONFigure:EVDO:SIGNaling<Instance>:CARRier:FLFRequency
CONFigure:EVDO:SIGNaling<Instance>:CARRier:RLFRequency
```

class Carrier

Carrier commands group definition. 6 total commands, 1 Sub-groups, 4 group commands

get_channel() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:CHANnel
value: int = driver.configure.carrier.get_channel()
```

Sets/gets the channel for a carrier in the sector implemented by the signaling application. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

return carrier_channel: The range of possible channels depends on the selected band class. The values below are for band class BC0 (US-Cellular) . For an overview, see 'Band Classes'. Range: 1 to 799, 991 to 1323

get_fl_frequency() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:FLFRequency
value: int = driver.configure.carrier.get_fl_frequency()
```

Gets the forward link frequency for a carrier in the cell implemented by the signaling application. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. This frequency is determined by the cell's main carrier channel and the related carrier's channel offset.

return cfwd_link_freq: Range: 100 MHz to 6.1 GHz

get_rl_frequency() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:RLFRequency
value: int = driver.configure.carrier.get_rl_frequency()
```

Gets the reverse link frequency for a carrier in the cell implemented by the signaling application. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. This frequency is determined by the cell's main carrier channel and the related carrier's channel offset.

return crev_link_freq: Range: 100 MHz to 6.1 GHz

get_setting() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:SETTing
value: int = driver.configure.carrier.get_setting()
```

Sets/gets the carrier to which subsequent carrier-related commands apply.

return set_carrier: Range: 0 to 2

set_channel(*carrier_channel: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:CHANnel
driver.configure.carrier.set_channel(carrier_channel = 1)
```

Sets/gets the channel for a carrier in the sector implemented by the signaling application. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

param carrier_channel The range of possible channels depends on the selected band class. The values below are for band class BC0 (US-Cellular) . For an overview, see 'Band Classes'. Range: 1 to 799, 991 to 1323

set_setting(*set_carrier: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:SETTing
driver.configure.carrier.set_setting(set_carrier = 1)
```

Sets/gets the carrier to which subsequent carrier-related commands apply.

param set_carrier Range: 0 to 2

Cloning the Group

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

Subgroups

7.1.5.1 Level

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:CARRier:LEVel:ABSolute
CONFigure:EVDO:SIGNaling<Instance>:CARRier:LEVel:RELative
```

class Level

Level commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_absolute() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:LEVel:ABSolute
value: int = driver.configure.carrier.level.get_absolute()
```

Sets/gets the absolute 1xEV-DO power level for a carrier. Preselect the related carrier using the method RsCmwEvdoSig. Configure.Carrier.setting command. While the query can be executed for all carriers, setting the power level is only possible for carrier 0. The power level of carriers 1 and 2 are specified via their level relative to carrier 0 - implicitly determining the absolute levels.

return level_absolute: Range: -180 dBm to 90 dBm, Unit: dBm

get_relative() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:LEVel:RELative
value: int = driver.configure.carrier.level.get_relative()
```

Sets/gets the relative 1xEV-DO power for a carrier in the sector implemented by the signaling application. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. While the query can be executed for all carriers (with return value 0 for carrier 0) , setting the relative 1xEV-DO power is only possible for carriers 1 and 2.

return level_relative: The level is relative to the main carrier's absolute power. Range: -20 dB to 0 dB, Unit: dB

set_absolute(*level_absolute: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:LEVel:ABSolute
driver.configure.carrier.level.set_absolute(level_absolute = 1)
```

Sets/gets the absolute 1xEV-DO power level for a carrier. Preselect the related carrier using the method RsCmwEvdoSig. Configure.Carrier.setting command. While the query can be executed for all carriers, setting the power level is only possible for carrier 0. The power level of carriers 1 and 2 are specified via their level relative to carrier 0 - implicitly determining the absolute levels.

param level_absolute Range: -180 dBm to 90 dBm, Unit: dBm

set_relative(*level_relative: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CARRier:LEVel:RELative
driver.configure.carrier.level.set_relative(level_relative = 1)
```

Sets/gets the relative 1xEV-DO power for a carrier in the sector implemented by the signaling application. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. While the query can be executed for all carriers (with return value 0 for carrier 0) , setting the relative 1xEV-DO power is only possible for carriers 1 and 2.

param level_relative The level is relative to the main carrier's absolute power. Range: -20 dB to 0 dB, Unit: dB

7.1.6 Sector

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:SECTor:SETTing
```

class Sector

Sector commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_setting() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SECTor:SETTing
value: int = driver.configure.sector.get_setting()
```

Sets/gets the sector to which subsequent sector-related commands apply.

return set_sector: Only sector 0 supported in the current release Range: 0 to 0

set_setting(*set_sector: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SECTor:SETTing
driver.configure.sector.set_setting(set_sector = 1)
```

Sets/gets the sector to which subsequent sector-related commands apply.

param set_sector Only sector 0 supported in the current release Range: 0 to 0

7.1.7 Pilot

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:PILot:SETTing
```

class Pilot

Pilot commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_setting() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:PILot:SETTing
value: int = driver.configure.pilot.get_setting()
```

Sets/gets the pilot to which subsequent pilot-related commands apply.

return set_pilot: Range: 0 to 2

set_setting(*set_pilot: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:PILot:SETTing
driver.configure.pilot.set_setting(set_pilot = 1)
```

Sets/gets the pilot to which subsequent pilot-related commands apply.

param set_pilot Range: 0 to 2

7.1.8 Cstatus

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:AFLCarriers
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:ARLCarriers
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:PLSubtype
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:IRAT
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:APPLication
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:UATI
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:ESN
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:MEID
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:EHRPd
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:LOG
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:ILCMask
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:QLCMask
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:MRBandwidth
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:MODE
```

class Cstatus

Cstatus commands group definition. 16 total commands, 1 Sub-groups, 14 group commands

get_afl_carriers() → RsCmwEvdoSig.enums.LinkCarrier

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:AFLCarriers
value: enums.LinkCarrier = driver.configure.cstatus.get_afl_carriers()
```

Queries the current state of a forward link carrier. Preselect the related carrier using the method RsCmwEvdoSig. Configure.Carrier.setting command.

INTRO_CMD_HELP: Note that a carrier can only be active on the AT if it is:

- Enabled on the cell (using method RsCmwEvdoSig.Configure.Network.Pilot.An.active)
- Assigned to the AT (using method RsCmwEvdoSig.Configure.Network.Pilot.At.assigned)

return act_fwd_link_carr: ACTive | NACTive | NCConnected | DISabled ACTive: The carrier is assigned to the AT and the traffic channel is active. NACTive: The carrier is assigned to the AT but the traffic channel is inactive. NCConnected: The carrier is assigned to the AT but the AT is not connected. DISabled: The carrier is not assigned to the AT.

get_application() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:APPLication
value: str = driver.configure.cstatus.get_application()
```

Returns the active test or packet applications along with the streams they are using.

return application: Comma-separated string of tuples s:Application Name, where s is the stream number, e.g. 1:FETAP

get_arl_carriers() → RsCmwEvdoSig.enums.LinkCarrier

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:ARLCarriers
value: enums.LinkCarrier = driver.configure.cstatus.get_arl_carriers()
```

Queries the current state of a reverse link carrier. Preselect the related carrier using the method RsCmwEvdoSig. Configure.Carrier.setting command.

INTRO_CMD_HELP: Note that a carrier can only be active on the AT if it is:

- Enabled on the cell (using method RsCmwEvdoSig.Configure.Network.Pilot.An.active)
- Assigned to the AT (using method RsCmwEvdoSig.Configure.Network.Pilot.At.assigned)

return act_rev_link_carr: ACTive | NACTive | NCConnected | DISabled ACTive: The carrier is assigned to the AT and the traffic channel is active. NACTive: The carrier is assigned to the AT but the traffic channel is inactive. NCConnected: The carrier is assigned to the AT but the AT is not connected. DISabled: The carrier is not assigned to the AT.

get_ehrpd() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:EHRPd
value: bool = driver.configure.cstatus.get_ehrpd()
```

Queries whether the AT supports eHRPD.

return enable: OFF | ON

get_esn() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:ESN
value: str = driver.configure.cstatus.get_esn()
```

Queries the electronic serial number of the connected AT.

return esn: 8-digit hexadecimal number Range: #H0 to #HFFFFFFFF (8 digits)

get_ilc_mask() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:ILCMask
value: str = driver.configure.cstatus.get_ilc_mask()
```

Queries the reverse traffic channel in phase long code mask associated with the access terminal's session.

return lc_mask_i: The long code mask in hexadecimal notation. Range: #H0 to #H3FFFFFFFFF

get_irat() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:IRAT
value: bool = driver.configure.cstatus.get_irat()
```

Indicates whether an inter-RAT handover is supported (as agreed during session negotiation) . Currently this command does not return valid results.

return inter_rat: OFF

get_log() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:LOG
value: str = driver.configure.cstatus.get_log()
```

Reports events and errors like connection state changes, RRC connection establishment/release and authentication failure.

return con_status_log: Report as a string

get_meid() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:MEID
value: str = driver.configure.cstatus.get_meid()
```

Queries the mobile equipment identifier (MEID) of the connected AT.

return meid: 14-digit hexadecimal number Range: #H0 to #HFFFFFFFFFFFFFF (14 digits)

get_mode() → RsCmwEvdoSig.enums.PrefAppMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:MODE
value: enums.PrefAppMode = driver.configure.cstatus.get_mode()
```

Queries the negotiated packet standard of the current connection.

return mode: EHRPd | HRPD Enhanced HRPD or high rate packet data (HRPD)

get_mr_bandwidth() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<Instance>:CSTatus:MRBandwidth
value: float = driver.configure.cstatus.get_mr_bandwidth()
```

Queries the maximum reverse link bandwidth reported by the AT.

return max_rev_bandwidth: Range: 0 Hz to 20 MHz, Unit: Hz

get_pl_subtype() → RsCmwEvdoSig.enums.PlSubtype

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:PLSubtype
value: enums.PlSubtype = driver.configure.cstatus.get_pl_subtype()
```

(continues on next page)

(continued from previous page)

```
    INTRO_CMD_HELP: Queries the active physical layer subtype, which in turn␣
↪depends on the selected network release (method RsCmwEvdoSig.Configure.
↪Network.release) :

    - With release 0 , the R&S CMW uses subtype 0
    - With revision A, the R&S CMW uses subtype 2
    - With revision B and more than 1 active carrier , the R&S CMW uses subtype␣
↪3; otherwise it uses subtype 2

    :return: pl_subtype: ST01 | ST2 | ST3 Physical layer subtype 0/1, 2 or 3.
```

get_qlcmask() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:QLCMask
value: str = driver.configure.cstatus.get_qlcmask()
```

Queries the reverse traffic channel quadrature-phase long code mask associated with the access terminal's session.

return lc_mask_q: The long code mask in hexadecimal notation. Range: #H0 to #H3FFFFFFFFF

get_uati() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:UATI
value: str = driver.configure.cstatus.get_uati()
```

Queries the unicast access terminal identifier (UATI) of the AT.

return uati: 8-digit hexadecimal number Range: #H0 to #HFFFFFFFF (8 digits)

Cloning the Group

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

Subgroups

7.1.8.1 PcChannel

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:PCCHannel:ENABle
CONFigure:EVDO:SIGNaling<Instance>:CSTatus:PCCHannel:CYCLe
```

class PcChannel

PcChannel commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_cycle() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:PCCHannel:CYCLe
value: int = driver.configure.cstatus.pcChannel.get_cycle()
```

Queries the control channel cycle in which the AT makes a transition out of the dormant state to monitor the control channel.

return cycle: Range: 0 to 32767 , Unit: (control channel cycles)

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CSTatus:PCCHannel:ENABle
value: bool = driver.configure.cstatus.pcChannel.get_enable()
```

Queries the state of the 'Preferred Control Channel Enable' flag. The flag indicates whether the AT selects the preferred control channel cycle.

return enable: OFF | ON

7.1.9 Mac

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:MAC:INDex
```

class Mac

Mac commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class IndexStruct

Structure for reading output parameters. Fields:

- Rev_0: int: Range: 5 to 63
- Rev_A: int: Range: 5 to 127
- Rev_B: int: Range: 5 to 127

get_index() → IndexStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:MAC:INDex
value: IndexStruct = driver.configure.mac.get_index()
```

Sets/gets a pilot's (associated carrier's) MAC index. Preselect the related pilot using the method RsCmwEvdoSig.Configure. Pilot.setting command.

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

set_index(*value: RsCmwEvdoSig.Implementations.Configure_.Mac.Mac.IndexStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:MAC:INDex
driver.configure.mac.set_index(value = IndexStruct())
```

Sets/gets a pilot's (associated carrier's) MAC index. Preselect the related pilot using the method RsCmwEvdoSig.Configure. Pilot.setting command.

param value see the help for IndexStruct structure arguments.

7.1.10 Layer

class Layer
Layer commands group definition. 61 total commands, 4 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.1 Connection

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:CONNection:ROMessages
CONFigure:EVDO:SIGNaling<Instance>:LAYer:CONNection:PDTHreshold
CONFigure:EVDO:SIGNaling<Instance>:LAYer:CONNection:RLFoffset
```

class Connection
Connection commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_pd_threshold() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:CONNection:PDTHreshold
value: float = driver.configure.layer.connection.get_pd_threshold()
```

Defines the pilot power (relative to the total 1xEV-DO power) below which the AT starts the pilot supervision timer and eventually announces that it has lost the network connection.

return pd_threshold: Range: -31.5 dB to 0 dB, Unit: dB

get_rlf_offset() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:CONNection:RLFoffset
value: int = driver.configure.layer.connection.get_rlf_offset()
```

Delays the reverse traffic data channel and reverse rate indicator channel (RRI) transmissions of the AT by an integer number of slots related to the system time-aligned frame boundary.

return rlf_offset: Range: 0 to 15

get_ro_messages() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:CONNection:ROMessages
value: bool = driver.configure.layer.connection.get_ro_messages()
```

Sets the 'redirect' bit in the QuickConfig message of the overhead messages protocol to '1'.

return ro_messages: OFF | ON

set_pd_threshold(*pd_threshold: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:CONNection:PDTHreshold
driver.configure.layer.connection.set_pd_threshold(pd_threshold = 1.0)
```

Defines the pilot power (relative to the total 1xEV-DO power) below which the AT starts the pilot supervision timer and eventually announces that it has lost the network connection.

param pd_threshold Range: -31.5 dB to 0 dB, Unit: dB

set_rlf_offset(*rlf_offset: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:CONNection:RLFoffset
driver.configure.layer.connection.set_rlf_offset(rlf_offset = 1)
```

Delays the reverse traffic data channel and reverse rate indicator channel (RRI) transmissions of the AT by an integer number of slots related to the system time-aligned frame boundary.

param rlf_offset Range: 0 to 15

set_ro_messages(*ro_messages: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:CONNection:ROMessages
driver.configure.layer.connection.set_ro_messages(ro_messages = False)
```

Sets the 'redirect' bit in the QuickConfig message of the overhead messages protocol to '1'.

param ro_messages OFF | ON

7.1.10.2 Application

class Application

Application commands group definition. 36 total commands, 7 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.2.1 Fmctap

class Fmctap

Fmctap commands group definition. 8 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.2.1.1 Drc

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:DRC:TYPE
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:DRC:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:DRC:SIZE
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:DRC:RATE
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:DRC:SLOTs
```

class Drc

Drc commands group definition. 5 total commands, 0 Sub-groups, 5 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:DRC:INDex
value: int = driver.configure.layer.application.fmctap.drc.get_index()
```

Queries the data rate index for the FMCTAP test packet stream on a particular carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. The DRC index is determined by the selected packet type (method RsCmwEvdoSig.Configure.Layer.Application.Fmctap.Drc.typePy) .

return drc_index: Range: 1 to 14

get_rate() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:DRC:RATE
value: float = driver.configure.layer.application.fmctap.drc.get_rate()
```

Queries the data rate of the FMCTAP test packet stream sent on a particular carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. The data rate is determined by the selected packet type (method RsCmwEvdoSig.Configure.Layer.Application.Fmctap.Drc.typePy) .

return drc_rate: Range: 4.8 kbit/s to 3072 kbit/s

get_size() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:DRC:SIZE
value: int = driver.configure.layer.application.fmctap.drc.get_size()
```

Queries the size of the FMCTAP test packets sent on a particular carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. The packet size is determined by the selected packet type (method RsCmwEvdoSig.Configure.Layer.Application.Fmctap.Drc.typePy) .

return drc_size: Range: 128 to 5120

get_slots() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:DRC:SLOTs
value: int = driver.configure.layer.application.fmctap.drc.get_slots()
```

Queries the slot count of an FMCTAP test packet sent on a particular carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. The slot count is determined by the selected packet type (method RsCmwEvdoSig.Configure.Layer.Application.Fmctap.Drc.typePy) .

return drc_slots: Range: 1 to 16

get_type_py() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:DRC:TYPE
value: int = driver.configure.layer.application.fmctap.drc.get_type_py()
```

Selects the type of FMCTAP test packets to be used. Preselect the related carrier using the method RsCmwEvdoSig.Configure. Carrier.setting command. Use the queries in the ...DRC... subsystem to query the corresponding DRC index, packet size, data rate, and slot count.

return drc_type: Range: 1 to 37

set_type_py(*drc_type: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:DRC:TYPE
driver.configure.layer.application.fmctap.drc.set_type_py(drc_type = 1)
```

Selects the type of FMCTAP test packets to be used. Preselect the related carrier using the method RsCmwEvdoSig.Configure. Carrier.setting command. Use the queries in the ...DRC... subsystem to query the corresponding DRC index, packet size, data rate, and slot count.

param drc_type Range: 1 to 37

7.1.10.2.1.2 Lback

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:LBACk:ENABle
```

class Lback

Lback commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:LBACk:ENABle
value: bool = driver.configure.layer.application.fmctap.lback.get_enable()
```

Queries whether the AT under test transmits FMCTAP loopback packets to measure packet error rate (PER) .

return lback: OFF | ON

7.1.10.2.1.3 Ack

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:ACK:FMODe
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FMCTap:ACK:MTYPe
```

class Ack

Ack commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_fmode() → RsCmwEvdoSig.enums.Fmode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:ACK:FMODe
value: enums.Fmode = driver.configure.layer.application.fmctap.ack.get_fmode()
```

Configures the ACK channel bit fixed mode (see 3GPP2 C.S0029) for a carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

return fmode: NUSed | AALWays | NAALways Not used, ACK always, NACK always

get_mtype() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:ACK:MTYPe
value: bool = driver.configure.layer.application.fmctap.ack.get_mtype()
```

Queries if the ACK channel modulation type fixed mode (see 3GPP2 C.S0029) is enabled for a carrier. Currently this mode is always switched off. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

return mtype: OFF | ON

set_fmode(*fmode: RsCmwEvdoSig.enums.Fmode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FMCTap:ACK:FMODe
driver.configure.layer.application.fmctap.ack.set_fmode(fmode = enums.Fmode.
↪AALWays)
```

Configures the ACK channel bit fixed mode (see 3GPP2 C.S0029) for a carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

param fmode NUSed | AALWays | NAALways Not used, ACK always, NACK always

7.1.10.2.2 Rmctap

class Rmctap

Rmctap commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.2.2.1 Smin

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RMCTap:SMIN:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RMCTap:SMIN:SIZE
```

class Smin

Smin commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RMCTap:SMIN:INDex
value: int = driver.configure.layer.application.rmctap.smin.get_index()
```

Selects the minimum packet size index for RMCTAP test packets on this carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. Use method RsCmwEvdoSig.Configure.Layer.Application.Retap.Smin. size to query the corresponding packet size.

return min_index: Range: 0 to 12

get_size() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RMCTap:SMIN:SIZE
value: int = driver.configure.layer.application.rmctap.smin.get_size()
```

Queries the minimum RMCTAP test packet size on a carrier. Preselect the related carrier using the method RsCmwEvdoSig. Configure.Carrier.setting command. The minimum packet size is determined by the selected minimum packet size index (method RsCmwEvdoSig.Configure.Layer.Application.Retap.Smin.index) .

return min_size: Range: 0 bits to 12.288E+3 bits

set_index(*min_index: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RMCTap:SMIN:INDex
driver.configure.layer.application.rmctap.smin.set_index(min_index = 1)
```

Selects the minimum packet size index for RMCTAP test packets on this carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. Use method RsCmwEvdoSig.Configure.Layer.Application.Retap.Smin. size to query the corresponding packet size.

param min_index Range: 0 to 12

7.1.10.2.2.2 Smax

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RMCTap:SMAX:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RMCTap:SMAX:SIZE
```

class Smax

Smax commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RMCTap:SMAX:INDex
value: int = driver.configure.layer.application.rmctap.smax.get_index()
```

Selects the maximum packet size index for RMCTAP test packets on this carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. Use method RsCmwEvdoSig.Configure.Layer.Application.Rmctap.Smax. size to query the corresponding packet size.

return max_index: Range: 1 to 12

get_size() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RMCTap:SMAX:SIZE
value: int = driver.configure.layer.application.rmctap.smax.get_size()
```

Queries the maximum RMCTAP test packet size on a carrier. Preselect the related carrier using the method RsCmwEvdoSig. Configure.Carrier.setting command. The maximum packet size is determined by the selected minimum packet size index (method RsCmwEvdoSig.Configure.Layer.Application.Retap.Smin.index) .

return max_size: Range: 0 bits to 12.288E+3 bits

set_index(*max_index: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RMCTap:SMAX:INDex
driver.configure.layer.application.rmctap.smax.set_index(max_index = 1)
```

Selects the maximum packet size index for RMCTAP test packets on this carrier. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. Use method RsCmwEvdoSig.Configure.Layer.Application.Rmctap.Smax. size to query the corresponding packet size.

param max_index Range: 1 to 12

7.1.10.2.3 Ftap

class Ftap

Ftap commands group definition. 5 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.2.3.1 Drc

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FTAP:DRC:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FTAP:DRC:RATE
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FTAP:DRC:SLOTs
```

class Drc

Drc commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FTAP:DRC:INDex
value: int = driver.configure.layer.application.ftap.drc.get_index()
```

Selects the data rate index for FTAP packets. Data rate index '0' stops the flow of FTAP packets to the AT. Use method RsCmwEvdoSig.Configure.Layer.Application.Ftap.Drc.rate and method RsCmwEvdoSig.Configure.Layer.Application.Ftap.Drc.slots to query the data rate and slot count for the selected index.

return drc_index: Range: 0 to 12

get_rate() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FTAP:DRC:RATE
value: float = driver.configure.layer.application.ftap.drc.get_rate()
```

Queries the data rate for the selected FTAP data rate index (method RsCmwEvdoSig.Configure.Layer.Application.Ftap.Drc. index) .

return drc_rate: Range: 0 kbit/s to 2457.6 kbit/s

get_slots() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FTAP:DRC:SLOTs
value: int = driver.configure.layer.application.ftap.drc.get_slots()
```

Queries the slot count for the selected FTAP data rate index (method RsCmwEvdoSig.Configure.Layer.Application.Ftap.Drc. index) .

return drc_slots: Range: 1 to 16

set_index(*drc_index: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FTAP:DRC:INDex
driver.configure.layer.application.ftap.drc.set_index(drc_index = 1)
```

Selects the data rate index for FTAP packets. Data rate index '0' stops the flow of FTAP packets to the AT. Use method RsCmwEvdoSig.Configure.Layer.Application.Ftap.Drc.rate and method RsCmwEvdoSig.Configure.Layer.Application.Ftap.Drc.slots to query the data rate and slot count for the selected index.

param drc_index Range: 0 to 12

7.1.10.2.3.2 Lback

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FTAP:LBACk:ENABle
```

class Lback

Lback commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FTAP:LBACk:ENABle
value: bool = driver.configure.layer.application.ftap.lback.get_enable()
```

Indicates whether the AT under test can transmit FTAP loopback packets to provide packet error rate (PER) information.

return lback: OFF | ON

7.1.10.2.3.3 Ack

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FTAP:ACK:FMODe
```

class Ack

Ack commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_fmode() → RsCmwEvdoSig.enums.Fmode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FTAP:ACK:FMODe
value: enums.Fmode = driver.configure.layer.application.ftap.ack.get_fmode()
```

Configures the ACK channel in the reverse signal that the AT uses for the acknowledgment of test packets received on the forward traffic channel.

return fmode: NUSed | AALWays | NAALways Not used, ACK always, NACK always

set_fmode(*fmode: RsCmwEvdoSig.enums.Fmode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FTAP:ACK:FMODe
driver.configure.layer.application.ftap.ack.set_fmode(fmode = enums.Fmode.
↪AALWays)
```

Configures the ACK channel in the reverse signal that the AT uses for the acknowledgment of test packets received on the forward traffic channel.

param fmode NUSed | AALWays | NAALways Not used, ACK always, NACK always

7.1.10.2.4 Rtap

class Rtap

Rtap commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.2.4.1 Rmin

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RTAP:RMIN:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RTAP:RMIN:RATE
```

class Rmin

Rmin commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RTAP:RMIN:INDex
value: int = driver.configure.layer.application.rtap.rmin.get_index()
```

Selects the minimum data rate index for RTAP packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Rtap.Rmin.rate to query the corresponding data rate.

return rmin_index: Range: 0 to 5

get_rate() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RTAP:RMIN:RATE
value: float = driver.configure.layer.application.rtap.rmin.get_rate()
```

Queries the data rate for the selected minimum data rate index (method RsCmwEvdoSig.Configure.Layer.Application.Rtap.Rmin. index) .

return rmin_rate: Range: 0 kbit/s to 153.6 kbit/s

set_index(*rmin_index: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RTAP:RMIN:INDex
driver.configure.layer.application.rtap.rmin.set_index(rmin_index = 1)
```

Selects the minimum data rate index for RTAP packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Rtap.Rmin.rate to query the corresponding data rate.

param rmin_index Range: 0 to 5

7.1.10.2.4.2 Rmax

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RTAP:RMAX:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RTAP:RMAX:RATE
```

class Rmax

Rmax commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RTAP:RMAX:INDex
value: int = driver.configure.layer.application.rtap.rmax.get_index()
```

Selects the maximum data rate index for RTAP packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Rtap.Rmax.rate to query the corresponding data rate.

return rmax_index: Range: 1 to 5

get_rate() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RTAP:RMAX:RATE
value: float = driver.configure.layer.application.rtap.rmax.get_rate()
```

Queries the data rate for the selected maximum data rate index (method RsCmwEvdoSig.Configure.Layer.Application.Rtap.Rmax. index) .

return rmax_rate: Range: 0 kbit/s to 153.6 kbit/s

set_index(*rmax_index: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RTAP:RMAX:INDex
driver.configure.layer.application.rtap.rmax.set_index(rmax_index = 1)
```

Selects the maximum data rate index for RTAP packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Rtap.Rmax.rate to query the corresponding data rate.

param rmax_index Range: 1 to 5

7.1.10.2.5 Fetap

class Fetap

Fetap commands group definition. 8 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.2.5.1 Drc

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:DRC:TYPE
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:DRC:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:DRC:SIZE
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:DRC:RATE
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:DRC:SLOTs
```

class Drc

Drc commands group definition. 5 total commands, 0 Sub-groups, 5 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:DRC:INDex
value: int = driver.configure.layer.application.fetap.drc.get_index()
```

Queries the data rate index for FETAP packets, depending on the selected packet type (method RsCmwEvdoSig.Configure.Layer. Application.Fetap.Drc.typePy) .

return drc_index: Range: 1 to 14

get_rate() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:DRC:RATE
value: float = driver.configure.layer.application.fetap.drc.get_rate()
```

Queries the data rate for FETAP packets, depending on the selected packet type (method RsCmwEvdoSig.Configure.Layer. Application.Fetap.Drc.typePy) .

return drc_rate: Range: 4.8 kbit/s to 3072 kbit/s

get_size() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:DRC:SIZE
value: int = driver.configure.layer.application.fetap.drc.get_size()
```

Queries the packet size for FETAP packets, depending on the selected packet type (method RsCmwEvdoSig.Configure.Layer. Application.Fetap.Drc.typePy) .

return drc_size: Range: 128 to 5120

get_slots() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:DRC:SLOTs
value: int = driver.configure.layer.application.fetap.drc.get_slots()
```

Queries the slot count for FETAP packets, depending on the selected packet type (method RsCmwEvdoSig.Configure.Layer. Application.Fetap.Drc.typePy) .

return drc_slots: Range: 1 to 16

get_type_py() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:DRC:TYPE
value: int = driver.configure.layer.application.fetap.drc.get_type_py()
```

Selects the packet type for FETAP packets. Use the queries in the ...DRC... subsystem to query the corresponding DRC index, packet size, data rate, and slot count.

return drc_type: Range: 1 to 37

set_type_py(*drc_type: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:DRC:TYPE
driver.configure.layer.application.fetap.drc.set_type_py(drc_type = 1)
```

Selects the packet type for FETAP packets. Use the queries in the ...DRC... subsystem to query the corresponding DRC index, packet size, data rate, and slot count.

param drc_type Range: 1 to 37

7.1.10.2.5.2 Lback

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:LBACk:ENABle
```

class Lback

Lback commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:LBACk:ENABle
value: bool = driver.configure.layer.application.fetap.lback.get_enable()
```

Indicates whether the AT under test can transmit FETAP loopback packets to provide packet error rate (PER) information.

return lback: OFF | ON

7.1.10.2.5.3 Ack

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:ACK:FMODe
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:FETap:ACK:MTYPe
```

class Ack

Ack commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_fmode() → RsCmwEvdoSig.enums.Fmode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:ACK:FMODe
value: enums.Fmode = driver.configure.layer.application.fetap.ack.get_fmode()
```

Configures the ACK channel in the reverse signal that the AT uses for the acknowledgment of test packets received on the forward traffic channel.

return fmode: NUSed | AALWays | NAALways Not used, ACK always, NACK always

get_mtype() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:ACK:MTYPe
value: bool = driver.configure.layer.application.fetap.ack.get_mtype()
```

Queries the status of the 'ACK channel modulation type fixed' mode where the AT can select a specific modulation type for the reverse ACK channel. In the current version, this mode is disabled.

return mtype: OFF

set_fmode(*fmode: RsCmwEvdoSig.enums.Fmode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:FETap:ACK:FMODe
driver.configure.layer.application.fetap.ack.set_fmode(fmode = enums.Fmode.
↪AALWays)
```

Configures the ACK channel in the reverse signal that the AT uses for the acknowledgment of test packets received on the forward traffic channel.

param fmode NUSed | AALWays | NAALways Not used, ACK always, NACK always

7.1.10.2.6 Retap

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RETap:TTARget
```

class Retap

Retap commands group definition. 5 total commands, 2 Sub-groups, 1 group commands

get_ttarget() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:TTARget
value: int = driver.configure.layer.application.retap.get_ttarget()
```

No command help available

return terminat_target: No help available

set_ttarget(*terminat_target: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:TTARget
driver.configure.layer.application.retap.set_ttarget(terminat_target = 1)
```

No command help available

param terminat_target No help available

Cloning the Group

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

Subgroups

7.1.10.2.6.1 Smin

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RETap:SMIN:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RETap:SMIN:SIZE
```

class Smin

Smin commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:SMIN:INDex
value: int = driver.configure.layer.application.retap.smin.get_index()
```

Selects the minimum packet size index for RETAP test packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Retap. Smin.size to query the corresponding packet size.

return min_index: Range: 0 to 12

get_size() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:SMIN:SIZE
value: int = driver.configure.layer.application.retap.smin.get_size()
```

Queries the minimum RETAP test packet size. This size is determined by the selected minimum packet size index (method RsCmwEvdoSig.Configure.Layer.Application.Retap.Smin.index) .

return min_size: Range: 0 bits to 12.288E+3 bits

set_index(*min_index: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:SMIN:INDex
driver.configure.layer.application.retap.smin.set_index(min_index = 1)
```

Selects the minimum packet size index for RETAP test packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Retap. Smin.size to query the corresponding packet size.

param min_index Range: 0 to 12

7.1.10.2.6.2 Smax

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RETap:SMAX:INDex
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:RETap:SMAX:SIZE
```

class Smax

Smax commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_index() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:SMAX:INDex
value: int = driver.configure.layer.application.retap.smax.get_index()
```

Selects the maximum data rate index for RETAP packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Retap.Smax. size to query the corresponding packet size.

return max_index: Range: 1 to 12

get_size() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:SMAX:SIZE
value: int = driver.configure.layer.application.retap.smax.get_size()
```

Queries the packet size for the selected maximum data rate index (method RsCmwEvdoSig.Configure.Layer.Application.Retap. Smax.index) .

return max_size: Range: 0 bits to 12.288E+3 bits

set_index(*max_index: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:RETap:SMAX:INDex
driver.configure.layer.application.retap.smax.set_index(max_index = 1)
```

Selects the maximum data rate index for RETAP packets. Use method RsCmwEvdoSig.Configure.Layer.Application.Retap.Smax. size to query the corresponding packet size.

param max_index Range: 1 to 12

7.1.10.2.7 Packet

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:PACKet:PREFerred
CONFigure:EVDO:SIGNaling<Instance>:LAYer:APPLication:PACKet:MODE
```

class Packet

Packet commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_mode() → RsCmwEvdoSig.enums.PrefAppMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:PACKet:MODE
value: enums.PrefAppMode = driver.configure.layer.application.packet.get_mode()
```

Sets the preferred standard to be signaled during the connection setup.

return pref_mode: EHRPd | HRPD Enhanced HRPD or high rate packet data (HRPD)

get_preferred() → RsCmwEvdoSig.enums.PrefApplication

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:PACKet:PREFerred
value: enums.PrefApplication = driver.configure.layer.application.packet.get_
↪preferred()
```

Selects the packet application the R&S CMW initially proposes to the AT during session negotiation. See 'Packet Applications' for details.

return pref_application: DPA | EMPA EMPA: Enhanced multi-flow packet application
DPA: Default packet application

set_mode(*pref_mode: RsCmwEvdoSig.enums.PrefAppMode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:PACKet:MODE
driver.configure.layer.application.packet.set_mode(pref_mode = enums.
↪PrefAppMode.EHRPd)
```

Sets the preferred standard to be signaled during the connection setup.

param pref_mode EHRPd | HRPD Enhanced HRPD or high rate packet data (HRPD)

set_preferred(*pref_application: RsCmwEvdoSig.enums.PrefApplication*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:APPLication:PACKet:PREFerred
driver.configure.layer.application.packet.set_preferred(pref_application =␣
↪enums.PrefApplication.DPA)
```

Selects the packet application the R&S CMW initially proposes to the AT during session negotiation. See 'Packet Applications' for details.

param pref_application DPA | EMPA EMPA: Enhanced multi-flow packet application
DPA: Default packet application

7.1.10.3 Mac

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:TTOPt
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DRATe
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:SSEed
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:MPSequences
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:IPBackoff
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:IPSBackoff
```

class Mac

Mac commands group definition. 20 total commands, 3 Sub-groups, 6 group commands

get_drate() → RsCmwEvdoSig.enums.CtrlChannelDataRate

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRATe
value: enums.CtrlChannelDataRate = driver.configure.layer.mac.get_drate()
```

Defines the data rate for asynchronous control channels.

return data_rate: R384 | R768

get_ip_backoff() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:IPBackoff
value: int = driver.configure.layer.mac.get_ip_backoff()
```

Defines the upper limit of the backoff range (in units of 'access cycle duration' defined in the 'Network' section) which the AT uses between access probes.

return ip_backoff: Range: 1 to 15

get_ips_backoff() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:IPSBackoff
value: int = driver.configure.layer.mac.get_ips_backoff()
```

Defines the upper limit of the backoff range in units of access cycle duration (defined in the 'Network' section) which the AT uses between access probe sequences.

return ips_backoff: Range: 1 to 15

get_mp_sequences() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:MPSequences
value: int = driver.configure.layer.mac.get_mp_sequences()
```

Specifies the maximum number of access probe sequences for a single access attempt.

return mp_sequences: Range: 1 to 15

get_sseed() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:SSEed
value: str = driver.configure.layer.mac.get_sseed()
```

Queries the session seed parameter which is negotiated between the R&S CMW and the AT.

return sseed: Range: 0 to 4.294967295E+9

get_ttopt() → RsCmwEvdoSig.enums.T2Pmode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:TTOPt
value: enums.T2Pmode = driver.configure.layer.mac.get_ttopt()
```

Sets the T2P values in the session negotiation regarding the test purpose.

return mode: TPUT | RFCO TPUT: Throughput optimized RFCO: RF conformance

set_drate(*data_rate: RsCmwEvdoSig.enums.CtrlChannelDataRate*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRATe
driver.configure.layer.mac.set_drate(data_rate = enums.CtrlChannelDataRate.R384)
```

Defines the data rate for asynchronous control channels.

param data_rate R384 | R768

set_ip_backoff(*ip_backoff: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:IPBackoff
driver.configure.layer.mac.set_ip_backoff(ip_backoff = 1)
```

Defines the upper limit of the backoff range (in units of 'access cycle duration' defined in the 'Network' section) which the AT uses between access probes.

param ip_backoff Range: 1 to 15

set_ips_backoff(*ips_backoff: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:IPSBackoff
driver.configure.layer.mac.set_ips_backoff(ips_backoff = 1)
```

Defines the upper limit of the backoff range in units of access cycle duration (defined in the 'Network' section) which the AT uses between access probe sequences.

param ips_backoff Range: 1 to 15

set_mp_sequences(*mp_sequences: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:MPSequences
driver.configure.layer.mac.set_mp_sequences(mp_sequences = 1)
```

Specifies the maximum number of access probe sequences for a single access attempt.

param mp_sequences Range: 1 to 15

set_ttopt(*mode: RsCmwEvdoSig.enums.T2Pmode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:TTOPt
driver.configure.layer.mac.set_ttopt(mode = enums.T2Pmode.RFCO)
```

Sets the T2P values in the session negotiation regarding the test purpose.

param mode TPUT | RFCO TPUT: Throughput optimized RFCO: RF conformance

Cloning the Group

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

Subgroups

7.1.10.3.1 EftProtocol

class EftProtocol

EftProtocol commands group definition. 6 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.3.1.1 Drc

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:EFTProtocol:DRC:COVer
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:EFTProtocol:DRC:LENGth
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:EFTProtocol:DRC:CGAin
```

class Drc

Drc commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_cgain() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DRC:CGAin
value: float = driver.configure.layer.mac.eftProtocol.drc.get_cgain()
```

Defines a carrier's power level ratio of the reverse data rate control channel relative to the reverse pilot channel (subtype 2 and 3 signals only) . Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

return drcc_gain: Range: -9 dB to 6 dB, Unit: dB

get_cover() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DRC:COVer
value: int = driver.configure.layer.mac.eftProtocol.drc.get_cover()
```

Specifies the DRC cover value that the AT is to use on its data rate control (DRC) channel (for subtype 2 and 3 signals only) .

return drc_cover: Range: 1 to 6

get_length() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DRC:LENGth
value: int = driver.configure.layer.mac.eftProtocol.drc.get_length()
```

Defines the number of slots that the AT uses to send a single DRC message on a carrier (for subtype 2 and 3 signals only) . Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

return drc_length: Range: 1 to 8, Unit: slots

set_cgain(*drcc_gain: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DRC:CGAin
driver.configure.layer.mac.eftProtocol.drc.set_cgain(drcc_gain = 1.0)
```

Defines a carrier's power level ratio of the reverse data rate control channel relative to the reverse pilot channel (subtype 2 and 3 signals only) . Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

param drcc_gain Range: -9 dB to 6 dB, Unit: dB

set_cover(*drc_cover: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DRC:COVer
driver.configure.layer.mac.eftProtocol.drc.set_cover(drc_cover = 1)
```

Specifies the DRC cover value that the AT is to use on its data rate control (DRC) channel (for subtype 2 and 3 signals only) .

param drc_cover Range: 1 to 6

set_length(*drc_length: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DRC:LENGth
driver.configure.layer.mac.eftProtocol.drc.set_length(drc_length = 1)
```

Defines the number of slots that the AT uses to send a single DRC message on a carrier (for subtype 2 and 3 signals only) . Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

param drc_length Range: 1 to 8, Unit: slots

7.1.10.3.1.2 Dsc

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:EFTProtocol:DSC:VALue
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:EFTProtocol:DSC:CGAin
```

class Dsc

Dsc commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_cgain() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DSC:CGAin
value: float = driver.configure.layer.mac.eftProtocol.dsc.get_cgain()
```

Sets the power of the reverse DSC relative to the power of the reverse pilot channel (for subtype 2 and 3 signals only) .

return dscch_gain: Range: -15.5 dB to 0 dB, Unit: dB

get_value() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DSC:VALue
value: int = driver.configure.layer.mac.eftProtocol.dsc.get_value()
```

Specifies the value that the AT is to use on the data source channel (DSC) to select the serving sector simulated by the R&S CMW (for subtype 2 and 3 signals only) .

return dsc_value: Range: 1 to 7

set_cgain(*dscch_gain: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DSC:CGAin
driver.configure.layer.mac.eftProtocol.dsc.set_cgain(dscch_gain = 1.0)
```

Sets the power of the reverse DSC relative to the power of the reverse pilot channel (for subtype 2 and 3 signals only) .

param dscch_gain Range: -15.5 dB to 0 dB, Unit: dB

set_value(*dsc_value: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:DSC:VALue
driver.configure.layer.mac.eftProtocol.dsc.set_value(dsc_value = 1)
```

Specifies the value that the AT is to use on the data source channel (DSC) to select the serving sector simulated by the R&S CMW (for subtype 2 and 3 signals only) .

param dsc_value Range: 1 to 7

7.1.10.3.1.3 Ack

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:EFTProtocol:ACK:CGAin
```

class Ack

Ack commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_cgain() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:ACK:CGAin
value: float = driver.configure.layer.mac.eftProtocol.ack.get_cgain()
```

Defines the ratio of the power of the reverse ACK channel to the power of the reverse pilot channel on a carrier (for subtype 2 and 3 signals only) . Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier. setting command.

return ackc_gain: Range: -3 dB to 6 dB, Unit: dB

set_cgain(*ackc_gain: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:EFTProtocol:ACK:CGAin
driver.configure.layer.mac.eftProtocol.ack.set_cgain(ackc_gain = 1.0)
```

Defines the ratio of the power of the reverse ACK channel to the power of the reverse pilot channel on a carrier (for subtype 2 and 3 signals only) . Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier. setting command.

param ackc_gain Range: -3 dB to 6 dB, Unit: dB

7.1.10.3.2 DftProtocol

class DftProtocol

DftProtocol commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.3.2.1 Drc

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DFTProtocol:DRC:COVer
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DFTProtocol:DRC:LENGth
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DFTProtocol:DRC:CGAin
```

class Drc

Drc commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_cgain() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:DRC:CGAin
value: float = driver.configure.layer.mac.dftProtocol.drc.get_cgain()
```

Defines the ratio of the power of the reverse data rate control channel to the power of the reverse pilot channel (for subtype 0/1 signals only) .

return drcc_gain: Range: -9 dB to 6 dB

get_cover() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:DRC:COVer
value: int = driver.configure.layer.mac.dftProtocol.drc.get_cover()
```

Specifies the DRC cover value that the AT is to use on its data rate control (DRC) channel (for subtype 0/1 signals only) .

return drc_cover: Range: 1 to 6

get_length() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:DRC:LENGth
value: int = driver.configure.layer.mac.dftProtocol.drc.get_length()
```

Defines the number of slots that the AT uses to send a single DRC (for subtype 0/1 signals only) .

return drc_length: Range: 1 slots to 8 slots

set_cgain(*drcc_gain: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:DRC:CGAin
driver.configure.layer.mac.dftProtocol.drc.set_cgain(drcc_gain = 1.0)
```

Defines the ratio of the power of the reverse data rate control channel to the power of the reverse pilot channel (for subtype 0/1 signals only) .

param drcc_gain Range: -9 dB to 6 dB

set_cover(*drc_cover: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:DRC:COVer
driver.configure.layer.mac.dftProtocol.drc.set_cover(drc_cover = 1)
```

Specifies the DRC cover value that the AT is to use on its data rate control (DRC) channel (for subtype 0/1 signals only) .

param drc_cover Range: 1 to 6

set_length(*drc_length: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:DRC:LENGth
driver.configure.layer.mac.dftProtocol.drc.set_length(drc_length = 1)
```

Defines the number of slots that the AT uses to send a single DRC (for subtype 0/1 signals only) .

param drc_length Range: 1 slots to 8 slots

7.1.10.3.2.2 Ack

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DFTProtocol:ACK:CGAin
```

class Ack

Ack commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_cgain() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:ACK:CGAin
value: float = driver.configure.layer.mac.dftProtocol.ack.get_cgain()
```

Defines the ratio of the power of the reverse ACK channel to the power of the reverse pilot channel (for subtype 0/1 signals only) .

return ackc_gain: Range: -3 dB to 6 dB, Unit: dB

set_cgain(*ackc_gain: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DFTProtocol:ACK:CGAin
driver.configure.layer.mac.dftProtocol.ack.set_cgain(ackc_gain = 1.0)
```

Defines the ratio of the power of the reverse ACK channel to the power of the reverse pilot channel (for subtype 0/1 signals only) .

param ackc_gain Range: -3 dB to 6 dB, Unit: dB

7.1.10.3.3 DrtProtocol

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DRTProtocol:DONom
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DRTProtocol:DRATe
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DRTProtocol:ITRansition
CONFigure:EVDO:SIGNaling<Instance>:LAYer:MAC:DRTProtocol:DTRansition
```

class DrtProtocol

DrtProtocol commands group definition. 4 total commands, 0 Sub-groups, 4 group commands

class DrateStruct
Structure for reading output parameters. Fields:

- R_9_K: float: Range: -2 dB to 1.75 dB, Unit: dB
- R_19_K: float: Range: -2 dB to 1.75 dB, Unit: dB
- R_38_K: float: Range: -2 dB to 1.75 dB, Unit: dB
- R_76_K: float: Range: -2 dB to 1.75 dB, Unit: dB
- R_153_K: float: Range: -2 dB to 1.75 dB, Unit: dB

class DtransitionStruct
Structure for reading output parameters. Fields:

- R_19_K: str: Range: #H00 to #HFF
- R_38_K: str: Range: #H00 to #HFF
- R_76_K: str: Range: #H00 to #HFF
- R_153_K: str: Range: #H00 to #HFF

class ItransitionStruct
Structure for reading output parameters. Fields:

- R_9_K: str: Range: #H00 to #HFF
- R_19_K: str: Range: #H00 to #HFF
- R_38_K: str: Range: #H00 to #HFF
- R_76_K: str: Range: #H00 to #HFF

get_donom() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRTProtocol:DONom
value: float = driver.configure.layer.mac.drtProtocol.get_donom()
```

Defines the nominal offset of the reverse traffic channel power from the reverse pilot channel power. In the current version, this parameter is not supported.

return data_offset_nom: Range: depending on test settings , Unit: dB

get_drate() → DrateStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRTProtocol:DRATe
value: DrateStruct = driver.configure.layer.mac.drtProtocol.get_drate()
```

Defines the ratio of the reverse traffic channel power at different data rates to the reverse pilot channel power.

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

get_dtransition() → DtransitionStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRTProtocol:DTRansition
value: DtransitionStruct = driver.configure.layer.mac.drtProtocol.get_
↪dtransition()
```

Defines the probability of the access terminal to decrease its transmission rate to the next lower data rate.

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

get_itransition() → ItransitionStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRTProtocol:ITRansition
value: ItransitionStruct = driver.configure.layer.mac.drtProtocol.get_
↪itransition()
```

Defines the probability of the access terminal to increase its transmission rate to the next higher data rate.

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

set_drate(*value: RsCmwEvdoSig.Implementations.Configure_.Layer_.Mac_.DrtProtocol.DrtProtocol.DrateStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRTProtocol:DRATe
driver.configure.layer.mac.drtProtocol.set_drate(value = DrateStruct())
```

Defines the ratio of the reverse traffic channel power at different data rates to the reverse pilot channel power.

param value see the help for DrateStruct structure arguments.

set_dtransition(*value: RsCmwEvdoSig.Implementations.Configure_.Layer_.Mac_.DrtProtocol.DrtProtocol.DtransitionStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRTProtocol:DTRansition
driver.configure.layer.mac.drtProtocol.set_dtransition(value =␣
↪DtransitionStruct())
```

Defines the probability of the access terminal to decrease its transmission rate to the next lower data rate.

param value see the help for DtransitionStruct structure arguments.

set_itransition(*value: RsCmwEvdoSig.Implementations.Configure_.Layer_.Mac_.DrtProtocol.DrtProtocol.ItransitionStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:MAC:DRTProtocol:ITRansition
driver.configure.layer.mac.drtProtocol.set_itransition(value =␣
↪ItransitionStruct())
```

Defines the probability of the access terminal to increase its transmission rate to the next higher data rate.

param value see the help for ItransitionStruct structure arguments.

7.1.10.4 Session

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:LAYer:SESSion:ISTimeout
CONFigure:EVDO:SIGNaling<Instance>:LAYer:SESSion:SNINcluded
```

class Session

Session commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_is_timeout() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:SESSion:ISTimeout
value: int = driver.configure.layer.session.get_is_timeout()
```

Specifies the time after which the AT, if it does not detect any traffic from the R&S CMW directed to it, closes the session.

return is_timeout: Range: 0 min to 65.535E+3 min

get_sn_included() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:SESSion:SNINcluded
value: bool = driver.configure.layer.session.get_sn_included()
```

Specifies whether the ATISubnetMask field and the UATI104 field are included in the UATI assignment message.

return sn_included: OFF | ON

set_is_timeout(*is_timeout: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:SESSion:ISTimeout
driver.configure.layer.session.set_is_timeout(is_timeout = 1)
```

Specifies the time after which the AT, if it does not detect any traffic from the R&S CMW directed to it, closes the session.

param is_timeout Range: 0 min to 65.535E+3 min

set_sn_included(*sn_included: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:LAYer:SESSion:SNINcluded
driver.configure.layer.session.set_sn_included(sn_included = False)
```

Specifies whether the ATISubnetMask field and the UATI104 field are included in the UATI assignment message.

param sn_included OFF | ON

7.1.11 Network

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SID
CONFigure:EVDO:SIGNaling<Instance>:NETWork:RELease
```

class Network

Network commands group definition. 30 total commands, 5 Sub-groups, 2 group commands

get_release() → RsCmwEvdoSig.enums.NetworkRelease

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:RELease
value: enums.NetworkRelease = driver.configure.network.get_release()
```

Selects the network release for the signaling tests.

return release: R0 | RA | RB Release 0, A or B

get_sid() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SID
value: int = driver.configure.network.get_sid()
```

Defines the 15-bit system ID that the R&S CMW broadcasts on its forward 1xEV-DO signal

return system_id: Range: 0 to 32767 (2^15 - 1)

set_release(*release: RsCmwEvdoSig.enums.NetworkRelease*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:RELease
driver.configure.network.set_release(release = enums.NetworkRelease.R0)
```

Selects the network release for the signaling tests.

param release R0 | RA | RB Release 0, A or B

set_sid(*system_id: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SID
driver.configure.network.set_sid(system_id = 1)
```

Defines the 15-bit system ID that the R&S CMW broadcasts on its forward 1xEV-DO signal

param system_id Range: 0 to 32767 (2^15 - 1)

Cloning the Group

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

Subgroups

7.1.11.1 Sector

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:PNOFfset
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:CLRCode
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:SMASk
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:CNTCode
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:FORMat
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:NPBits
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:IDOVerall
```

class Sector

Sector commands group definition. 9 total commands, 1 Sub-groups, 7 group commands

get_clr_code() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:CLRCode
value: int = driver.configure.network.sector.get_clr_code()
```

Defines the 8-bit color code of the sector.

return clr_code: Range: 0 to 255

get_cnt_code() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:CNTCode
value: int = driver.configure.network.sector.get_cnt_code()
```

Defines the 3-digit decimal representation of the country code associated with the sector.

return cnt_code: Range: 0 to 999

get_format_py() → RsCmwEvdoSig.enums.SectorIdFormat

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:FORMat
value: enums.SectorIdFormat = driver.configure.network.sector.get_format_py()
```

Selects the input format for the 128-bit overall sector ID.

return format_py: A41N | MANual ANSI-41 or manual entry

get_id_overall() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:IDOVerall
value: str = driver.configure.network.sector.get_id_overall()
```

Queries the 128-bit overall ID of the sector in ANSI-41 format (method RsCmwEvdoSig.Configure.Network.Sector.formatPy) .

return ansi_41_overall_id: 32-digit hexadecimal number

get_npbits() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:NPBits
value: int = driver.configure.network.sector.get_npbits()
```

Defines the number of parity bits, to be used if the ANSI-41 format is selected (method RsCmwEvdoSig.Configure.Network. Sector.formatPy) .

return ansi_41_pbits: Range: 1 to 64

get_pn_offset() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:PNOFfset
value: int = driver.configure.network.sector.get_pn_offset()
```

Defines the pilot PN offset index of the generated forward 1xEV-DO signal.

return pn_offset: Range: 0 to 511

get_smask() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:SMASk
value: int = driver.configure.network.sector.get_smask()
```

Defines the 8-bit sector subnet identifier.

return smask: Range: 0 to 128

set_clr_code(*clr_code: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:CLRCode
driver.configure.network.sector.set_clr_code(clr_code = 1)
```

Defines the 8-bit color code of the sector.

param clr_code Range: 0 to 255

set_cnt_code(*cnt_code: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:CNTCode
driver.configure.network.sector.set_cnt_code(cnt_code = 1)
```

Defines the 3-digit decimal representation of the country code associated with the sector.

param cnt_code Range: 0 to 999

set_format_py(*format_py: RsCmwEvdoSig.enums.SectorIdFormat*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:FORMat
driver.configure.network.sector.set_format_py(format_py = enums.SectorIdFormat.
↪A41N)
```

Selects the input format for the 128-bit overall sector ID.

param format_py A41N | MANual ANSI-41 or manual entry

set_npbits(*ansi_41_pbits: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:NPBits
driver.configure.network.sector.set_npbits(ansi_41_pbits = 1)
```

Defines the number of parity bits, to be used if the ANSI-41 format is selected (method RsCmwEvdoSig.Configure.Network. Sector.formatPy) .

param ansi_41_pbits Range: 1 to 64

set_pn_offset(*pn_offset: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:PNOFfset
driver.configure.network.sector.set_pn_offset(pn_offset = 1)
```

Defines the pilot PN offset index of the generated forward 1xEV-DO signal.

param pn_offset Range: 0 to 511

set_smask(*smask: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:SMASk
driver.configure.network.sector.set_smask(smask = 1)
```

Defines the 8-bit sector subnet identifier.

param smask Range: 0 to 128

Cloning the Group

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

Subgroups

7.1.11.1.1 Id

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:ID:ANSI
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECTor:ID:MANual
```

class Id

Id commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_ansi() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:ID:ANSI
value: float = driver.configure.network.sector.id.get_ansi()
```

Defines the 24-bit sector ID, to be used if the ANSI-41 format is selected (method RsCmwEvdoSig.Configure.Network.Sector. formatPy) .

return ansi_41_sector_id: Sector ID, 6-digit hexadecimal number Range: #H000000 to #HFFFFFF

get_manual() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:ID:MANual
value: float = driver.configure.network.sector.id.get_manual()
```

Defines the 128-bit overall ID of the sector, to be used if the manual format is selected (method RsCmwEvdoSig.Configure. Network.Sector.formatPy) .

return manual_sector_id: Sector ID, 32-digit hexadecimal number Range: #H00000000000000000000000000 to #HFFFFFFFFFFFFFFFFFFFFFFFFF

set_ansi(*ansi_41_sector_id: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:ID:ANSI
driver.configure.network.sector.id.set_ansi(ansi_41_sector_id = 1.0)
```

Defines the 24-bit sector ID, to be used if the ANSI-41 format is selected (method RsCmwEvdoSig.Configure.Network.Sector. formatPy) .

param ansi_41_sector_id Sector ID, 6-digit hexadecimal number Range: #H000000 to #HFFFFFF

set_manual(*manual_sector_id: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECTor:ID:MANual
driver.configure.network.sector.id.set_manual(manual_sector_id = 1.0)
```

Defines the 128-bit overall ID of the sector, to be used if the manual format is selected (method RsCmwEvdoSig.Configure. Network.Sector.formatPy) .

param manual_sector_id Sector ID, 32-digit hexadecimal number Range: #H00000000000000000000000000 to #HFFFFFFFFFFFFFFFFFFFFFFFFF

7.1.11.2 Pilot

class Pilot

Pilot commands group definition. 3 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.11.2.1 An

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:PILot:AN:ACTive
```

class An

An commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_active() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PILot:AN:ACTive
value: bool = driver.configure.network.pilot.an.get_active()
```

Sets/gets the state of a pilot in the cell implemented by the signaling application. Preselect the related pilot using the method RsCmwEvdoSig.Configure.Pilot.setting command.

return active_on_an: OFF | ON When set to OFF, the related carrier is physically disabled on the cell. Note that pilot 0 cannot be turned OFF.

set_active(*active_on_an: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PILot:AN:ACTive
driver.configure.network.pilot.an.set_active(active_on_an = False)
```

Sets/gets the state of a pilot in the cell implemented by the signaling application. Preselect the related pilot using the method RsCmwEvdoSig.Configure.Pilot.setting command.

param active_on_an OFF | ON When set to OFF, the related carrier is physically disabled on the cell. Note that pilot 0 cannot be turned OFF.

7.1.11.2.2 At

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:PILot:AT:ASSigned
CONFigure:EVDO:SIGNaling<Instance>:NETWork:PILot:AT:ACQuired
```

class At

At commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_acquired() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PILot:AT:ACQuired
value: bool = driver.configure.network.pilot.at.get_acquired()
```

Queries if a pilot is being acquired by the AT, i.e. if the corresponding carrier is carrying traffic. Preselect the related pilot using the method RsCmwEvdoSig.Configure.Pilot.setting command.

INTRO_CMD_HELP: Note that a pilot can only be acquired by the AT if it is:

- Activated on the cell (using method RsCmwEvdoSig.Configure.Network.Pilot.An.active)
- Assigned to the AT (using method RsCmwEvdoSig.Configure.Network.Pilot.At.assigned)

Use method RsCmwEvdoSig.Configure.Cstatus.aflCarriers to obtain more information on the carrier states.

return acquired_by_at: OFF | ON

get_assigned() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PILot:AT:ASSigned
value: bool = driver.configure.network.pilot.at.get_assigned()
```

Sets/gets the assignment state of a pilot. Assigning a pilot to the AT adds the pilot to the AT's active set (managed by the AN via TrafficChannelAssignment messages) . Preselect the related pilot using the method RsCmwEvdoSig.Configure.Pilot. setting command.

return assigned_to_at: OFF | ON A pilot can only be assigned to the AT if it is activated on the AN (see method RsCmwEvdoSig.Configure.Network.Pilot.An.active) . Note that pilot 0 cannot be unassigned. ON for pilot 0, OFF for pilots 1 and 2

set_assigned(*assigned_to_at: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PILot:AT:ASSigned
driver.configure.network.pilot.at.set_assigned(assigned_to_at = False)
```

Sets/gets the assignment state of a pilot. Assigning a pilot to the AT adds the pilot to the AT's active set (managed by the AN via TrafficChannelAssignment messages) . Preselect the related pilot using the method RsCmwEvdoSig.Configure.Pilot. setting command.

param assigned_to_at OFF | ON A pilot can only be assigned to the AT if it is activated on the AN (see method RsCmwEvdoSig.Configure.Network.Pilot.An.active) . Note that pilot 0 cannot be unassigned. ON for pilot 0, OFF for pilots 1 and 2

7.1.11.3 PropertyPy

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:PROPerty:CLDTime
CONFigure:EVDO:SIGNaling<Instance>:NETWork:PROPerty:FPACtivity
CONFigure:EVDO:SIGNaling<Instance>:NETWork:PROPerty:IRAT
```

class PropertyPy

PropertyPy commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_cld_time() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PROPerty:CLDTime
value: float or bool = driver.configure.network.propertyPy.get_cld_time()
```

Defines the time in s after which a connection is considered to be lost.

return cld_time: Range: 2 s to 6 s, Unit: s Additional OFF/ON disables/enables the call loss detect timer

get_fpactivity() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PROPerty:FPACtivity
value: int = driver.configure.network.propertyPy.get_fpactivity()
```

Defines the percentage of forward packets that the R&S CMW directs to the AT under test.

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

get_irat() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PROPerty:IRAT
value: bool = driver.configure.network.propertyPy.get_irat()
```

Flag for inter-RAT operability.

return inter_rat: OFF | ON

set_cld_time(*cld_time: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PROPerty:CLDTime
driver.configure.network.propertyPy.set_cld_time(cld_time = 1.0)
```

Defines the time in s after which a connection is considered to be lost.

param cld_time Range: 2 s to 6 s, Unit: s Additional OFF/ON disables/enables the call loss detect timer

set_fpactivity(*activity: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PROPerty:FPACtivity
driver.configure.network.propertyPy.set_fpactivity(activity = 1)
```

Defines the percentage of forward packets that the R&S CMW directs to the AT under test.

param activity Range: 0 % to 100 %, Unit: %

set_irat(*inter_rat: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:PROPerty:IRAT
driver.configure.network.propertyPy.set_irat(inter_rat = False)
```

Flag for inter-RAT operability.

param inter_rat OFF | ON

7.1.11.4 Aprobes

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:MODE
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:IADJust
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:OLADjust
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:PINCrement
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:PPSequence
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:PLENgth
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:ACDuration
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:PLSLots
CONFigure:EVDO:SIGNaling<Instance>:NETWork:APRobes:SAMRate
```

class Aprobes

Aprobes commands group definition. 9 total commands, 0 Sub-groups, 9 group commands

get_ac_duration() → RsCmwEvdoSig.enums.AccessDuration

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:ACDuration
value: enums.AccessDuration = driver.configure.network.aprobes.get_ac_duration()
```

Defines the length in slots of the access cycle.

return ac_duration: S16 | S32 | S64 | S128 16/32/64/128 slot cycle

get_iadjust() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:IADJust
value: int = driver.configure.network.aprobes.get_iadjust()
```

Specifies the initial power offset for access probes (INIT_PWR parameter in the access parameters message) .

return iadjust: Range: -16 dB to 15 dB, Unit: dB

get_mode() → RsCmwEvdoSig.enums.ProbesAckMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:MODE
value: enums.ProbesAckMode = driver.configure.network.aprobes.get_mode()
```

Specifies whether the tester acknowledges or ignores access probes from the AT.

return mode: ACKN | IGN

get_ol_adjust() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:OLADjust
value: int = driver.configure.network.aprobes.get_ol_adjust()
```

Specifies the nominal transmit power offset (NOM_PWR) to be used by ATs for the given band class in the open loop power estimate.

return ol_adjust: Range: -81 dB to -66 dB , Unit: dB

get_pincrement() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PINCrement
value: float = driver.configure.network.aprobes.get_pincrement()
```

Defines the step size of power increases (PWR_STEP) between consecutive access probes.

return pincrement: Range: 0 dB to 7.5 dB, Unit: dB

get_pl_slots() → RsCmwEvdoSig.enums.PlSlots

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PLSLots
value: enums.PlSlots = driver.configure.network.aprobes.get_pl_slots()
```

Defines the length in slots of the access probe preamble.

return pl_slots: S4 | S16 4/16 slot preamble

get_plength() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PLENgth
value: int = driver.configure.network.aprobes.get_plength()
```

Defines the length in frames of the access probe preamble.

return plength: Range: 1 frame to 6 frames

get_pp_sequence() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PPSequence
value: int = driver.configure.network.aprobes.get_pp_sequence()
```

Defines the maximum number of access probes which ATs are to transmit in a single access probe sequence.

return pp_sequence: Range: 1 to 15

get_sam_rate() → RsCmwEvdoSig.enums.SamRate

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:SAMRate
value: enums.SamRate = driver.configure.network.aprobes.get_sam_rate()
```

Defines the sector access maximum rate at which the AT can transmit on the access channel.

return sam_rate: R9K | R19K | R38K 9.6/19.2/38.4 kbit/s

set_ac_duration(*ac_duration: RsCmwEvdoSig.enums.AccessDuration*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:ACDuration
driver.configure.network.aprobes.set_ac_duration(ac_duration = enums.
↪AccessDuration.S128)
```

Defines the length in slots of the access cycle.

param ac_duration S16 | S32 | S64 | S128 16/32/64/128 slot cycle

set_iadjust(*iadjust: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:IADJust
driver.configure.network.aprobes.set_iadjust(iadjust = 1)
```

Specifies the initial power offset for access probes (INIT_PWR parameter in the access parameters message) .

param iadjust Range: -16 dB to 15 dB, Unit: dB

set_mode(*mode: RsCmwEvdoSig.enums.ProbesAckMode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:MODE
driver.configure.network.aprobes.set_mode(mode = enums.ProbesAckMode.ACKN)
```

Specifies whether the tester acknowledges or ignores access probes from the AT.

param mode ACKN | IGN

set_ol_adjust(*ol_adjust: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:OLADjust
driver.configure.network.aprobes.set_ol_adjust(ol_adjust = 1)
```

Specifies the nominal transmit power offset (NOM_PWR) to be used by ATs for the given band class in the open loop power estimate.

param ol_adjust Range: -81 dB to -66 dB , Unit: dB

set_pincrement(*pincrement: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PINCrement
driver.configure.network.aprobes.set_pincrement(pincrement = 1.0)
```

Defines the step size of power increases (PWR_STEP) between consecutive access probes.

param pincrement Range: 0 dB to 7.5 dB, Unit: dB

set_pl_slots(*pl_slots: RsCmwEvdoSig.enums.PlSlots*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PLSLots
driver.configure.network.aprobes.set_pl_slots(pl_slots = enums.PlSlots.S16)
```

Defines the length in slots of the access probe preamble.

param pl_slots S4 | S16 4/16 slot preamble

set_plength(*plength: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PLENgth
driver.configure.network.aprobes.set_plength(plength = 1)
```

Defines the length in frames of the access probe preamble.

param plength Range: 1 frame to 6 frames

set_pp_sequence(*pp_sequence: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:PPSequence
driver.configure.network.aprobes.set_pp_sequence(pp_sequence = 1)
```

Defines the maximum number of access probes which ATs are to transmit in a single access probe sequence.

param pp_sequence Range: 1 to 15

set_sam_rate(*sam_rate: RsCmwEvdoSig.enums.SamRate*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:APRobes:SAMRate
driver.configure.network.aprobes.set_sam_rate(sam_rate = enums.SamRate.R19K)
```

Defines the sector access maximum rate at which the AT can transmit on the access channel.

param sam_rate R9K | R19K | R38K 9.6/19.2/38.4 kbit/s

7.1.11.5 Security

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECurity:SKEY
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECurity:OPC
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECurity:AUTHenticat
CONFigure:EVDO:SIGNaling<Instance>:NETWork:SECurity:SQN
```

class Security

Security commands group definition. 4 total commands, 0 Sub-groups, 4 group commands

get_authenticate() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:AUTHenticat
value: str = driver.configure.network.security.get_authenticate()
```

Sets/gets the authentication management field (AMF) of the EAP-AKA' access authentication.

return authentication: 4 hexadecimal digits

get_opc() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:OPC
value: str = driver.configure.network.security.get_opc()
```

Sets/gets the operator variant Key (OPC) of the EAP-AKA' access authentication.

return operator_var_key: 32 hexadecimal digits

get_skey() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:SKEY
value: str = driver.configure.network.security.get_skey()
```

Sets/gets the shared authentication key of the EAP-AKA' access authentication.

return secret_key: 32 hexadecimal digits

get_sqn() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:SQN
value: str = driver.configure.network.security.get_sqn()
```

Sets/gets the sequence number sent to the AT in the EAP-request / AKA'-challenge message.

return sequence_number: 12 hexadecimal digits

set_authenticate(*authentication: str*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:AUTHenticat
driver.configure.network.security.set_authenticate(authentication = r1)
```

Sets/gets the authentication management field (AMF) of the EAP-AKA' access authentication.

param authentication 4 hexadecimal digits

set_opc(*operator_var_key: str*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:OPC
driver.configure.network.security.set_opc(operator_var_key = r1)
```

Sets/gets the operator variant Key (OPC) of the EAP-AKA' access authentication.

param operator_var_key 32 hexadecimal digits

set_skey(*secret_key: str*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:SKEY
driver.configure.network.security.set_skey(secret_key = r1)
```

Sets/gets the shared authentication key of the EAP-AKA' access authentication.

param secret_key 32 hexadecimal digits

set_sqn(*sequence_number: str*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NETWork:SECurity:SQN
driver.configure.network.security.set_sqn(sequence_number = r1)
```

Sets/gets the sequence number sent to the AT in the EAP-request / AKA'-challenge message.

param sequence_number 12 hexadecimal digits

7.1.12 Handoff

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:HANDoff:BCLass
CONFigure:EVDO:SIGNaling<Instance>:HANDoff:CHANnel
```

class Handoff

Handoff commands group definition. 7 total commands, 2 Sub-groups, 2 group commands

get_bclass() → RsCmwEvdoSig.enums.BandClass

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:BCLass
value: enums.BandClass = driver.configure.handoff.get_bclass()
```

Selects a handoff destination band class/network; see 'Band Classes'.

return band_class: USC | KCEL | NAPC | TACS | JTAC | KPCS | N45T | IM2K | NA7C | B18M | NA9C | NA8S | PA4M | PA8M | IEXT | USPC | AWS | U25B | U25F | PS7C | LO7C | LBANd | SBANd USC: BC 0, US-Cellular KCEL: BC 0, Korean Cellular NAPC: BC 1, North American PCS TACS: BC 2, TACS Band JTAC: BC 3, JTACS Band KPCS: BC 4, Korean PCS N45T: BC 5, NMT-450 IM2K: BC 6, IMT-2000 NA7C: BC 7, Upper 700 MHz B18M: BC 8, 1800 MHz Band NA9C: BC 9, North American 900 MHz NA8S: BC 10, Secondary 800 MHz PA4M: BC 11, European 400 MHz PAMR PA8M: BC 12, 800 MHz PAMR IEXT: BC 13, IMT-2000 2.5 GHz Extension USPC: BC 14, US PCS 1900 MHz AWS: BC 15, AWS Band U25B: BC 16, US 2.5 GHz Band PS7C: BC 18, Public Safety Band 700 MHz LO7C: BC 19, Lower 700 MHz LBAN: BC 20, L-Band SBAN: BC 21, S-Band

get_channel() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:CHANnel
value: int = driver.configure.handoff.get_channel()
```

Sets/gets the main RF channel (the only one for network releases 0/A) in the handoff destination cell.

return channel: The reset value and the range of possible channels depend on the selected band class; for an overview see 'Band Classes'. The values below are for band class BC0 (US Cellular) . Range: 1 to 799, 991 to 1323

set_bclass(*band_class: RsCmwEvdoSig.enums.BandClass*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:BCLass
driver.configure.handoff.set_bclass(band_class = enums.BandClass.AWS)
```

Selects a handoff destination band class/network; see 'Band Classes'.

param band_class USC | KCEL | NAPC | TACS | JTAC | KPCS | N45T | IM2K | NA7C | B18M | NA9C | NA8S | PA4M | PA8M | IEXT | USPC | AWS | U25B | U25F | PS7C | LO7C | LBANd | SBANd USC: BC 0, US-Cellular KCEL: BC 0, Korean Cellular NAPC: BC 1, North American PCS TACS: BC 2, TACS Band JTAC: BC 3, JTACS Band KPCS: BC 4, Korean PCS N45T: BC 5, NMT-450 IM2K: BC 6, IMT-2000 NA7C: BC 7, Upper 700 MHz B18M: BC 8, 1800 MHz Band NA9C: BC 9, North American 900 MHz NA8S: BC 10, Secondary 800 MHz PA4M: BC 11, European 400 MHz PAMR PA8M: BC 12, 800 MHz PAMR IEXT: BC 13, IMT-2000 2.5 GHz Extension USPC: BC 14, US PCS 1900 MHz AWS: BC 15, AWS Band U25B: BC 16, US 2.5 GHz Band PS7C: BC 18, Public Safety Band 700 MHz LO7C: BC 19, Lower 700 MHz LBAN: BC 20, L-Band SBAN: BC 21, S-Band

set_channel(*channel: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:CHANnel
driver.configure.handoff.set_channel(channel = 1)
```

Sets/gets the main RF channel (the only one for network releases 0/A) in the handoff destination cell.

param channel The reset value and the range of possible channels depend on the selected band class; for an overview see 'Band Classes'. The values below are for band class BC0 (US Cellular) . Range: 1 to 799, 991 to 1323

Cloning the Group

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

Subgroups

7.1.12.1 Carrier

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:HANDoff:CARRier:CHANnel
CONFigure:EVDO:SIGNaling<Instance>:HANDoff:CARRier:FLFRequency
CONFigure:EVDO:SIGNaling<Instance>:HANDoff:CARRier:RLFRequency
```

class Carrier

Carrier commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_channel() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:CARRier:CHANnel
value: int = driver.configure.handoff.carrier.get_channel()
```

Sets/gets the channel for a carrier in the handoff destination cell. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

return carrier_channel: The range of possible channels depends on the destination cell's selected band class. For an overview, see 'Band Classes'. The values below are for band class BC0 (US Cellular) . Range: 1 to 799, 991 to 1323

get_fl_frequency() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:CARRier:FLFRequency
value: int = driver.configure.handoff.carrier.get_fl_frequency()
```

Gets the forward link frequency for a carrier in the handoff destination cell. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. This frequency is determined by the handoff destination cell's main carrier channel and the related carrier's channel offset.

return cfwd_link_freq: Range: 100 MHz to 6.1 GHz

get_rl_frequency() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:CARRier:RLFRequency
value: int = driver.configure.handoff.carrier.get_rl_frequency()
```

Gets the reverse link frequency for a carrier in the handoff destination cell. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. This frequency is determined by the handoff destination cell's main carrier channel and the related carrier's channel offset.

return crev_link_freq: Range: 100 MHz to 6.1 GHz

set_channel(*carrier_channel: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:CARRier:CHANnel
driver.configure.handoff.carrier.set_channel(carrier_channel = 1)
```

Sets/gets the channel for a carrier in the handoff destination cell. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command.

param carrier_channel The range of possible channels depends on the destination cell's selected band class. For an overview, see 'Band Classes'. The values below are for band class BC0 (US Cellular) . Range: 1 to 799, 991 to 1323

7.1.12.2 Network

class Network

Network commands group definition. 2 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.12.2.1 Pilot

class Pilot
Pilot commands group definition. 2 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.12.2.1.1 An

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:HANDoff:NETWork:PILot:AN:ACTive
```

class An
An commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_active() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:NETWork:PILot:AN:ACTive
value: bool = driver.configure.handoff.network.pilot.an.get_active()
```

Sets/gets the state of a pilot in the handoff destination cell. Preselect the related pilot using the method RsCmwEvdoSig. Configure.Pilot.setting command.

return active_on_an: OFF | ON When set to OFF, the related carrier is physically disabled on the handoff destination cell. Note that pilot 0 cannot be turned OFF.

set_active(*active_on_an: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:NETWork:PILot:AN:ACTive
driver.configure.handoff.network.pilot.an.set_active(active_on_an = False)
```

Sets/gets the state of a pilot in the handoff destination cell. Preselect the related pilot using the method RsCmwEvdoSig. Configure.Pilot.setting command.

param active_on_an OFF | ON When set to OFF, the related carrier is physically disabled on the handoff destination cell. Note that pilot 0 cannot be turned OFF.

7.1.12.2.1.2 At

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:HANDoff:NETWork:PILot:AT:ASSigned
```

class At

At commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_assigned() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:NETWork:PILot:AT:ASSigned
value: bool = driver.configure.handoff.network.pilot.at.get_assigned()
```

Sets/gets the assignment state of a pilot in the handoff destination cell. Assigning a pilot to the AT adds the pilot to the AT's active set in the destination cell. Preselect the related pilot using the method RsCmwEvdoSig.Configure.Pilot. setting command.

return assigned_to_at: OFF | ON A pilot can only be assigned to the AT if it is activated on the destination cell (see method RsCmwEvdoSig.Configure.Handoff.Network.Pilot.An.active) . Note that pilot 0 cannot be unassigned.

set_assigned(*assigned_to_at: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:HANDoff:NETWork:PILot:AT:ASSigned
driver.configure.handoff.network.pilot.at.set_assigned(assigned_to_at = False)
```

Sets/gets the assignment state of a pilot in the handoff destination cell. Assigning a pilot to the AT adds the pilot to the AT's active set in the destination cell. Preselect the related pilot using the method RsCmwEvdoSig.Configure.Pilot. setting command.

param assigned_to_at OFF | ON A pilot can only be assigned to the AT if it is activated on the destination cell (see method RsCmwEvdoSig.Configure.Handoff.Network.Pilot.An.active) . Note that pilot 0 cannot be unassigned.

7.1.13 Mmonitor

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:MMONitor:ENABle
```

class Mmonitor

Mmonitor commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:MMONitor:ENABle
value: bool = driver.configure.mmonitor.get_enable()
```

Enable/disable message monitor

return enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:MMONitor:ENABle
driver.configure.mmonitor.set_enable(enable = False)
```

Enable/disable message monitor

param enable OFF | ON

Cloning the Group

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

Subgroups

7.1.13.1 IpAddress

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:MMONitor:IPADdress
```

class IpAddress

IpAddress commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class GetStruct

Response structure. Fields:

- Index: enums.IpAddressIndex: IP1 | IP2 | IP3
- Ip_Address: str: No parameter help available

get() → GetStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:MMONitor:IPADdress
value: GetStruct = driver.configure.mmonitor.ipAddress.get()
```

Select/get the target IP address for message monitoring (method RsCmwEvdoSig.Configure.Mmonitor.enable) . The IP addresses are centrally managed from the 'Setup' dialog.

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

set(*index: RsCmwEvdoSig.enums.IpAddressIndex*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:MMONitor:IPADdress
driver.configure.mmonitor.ipAddress.set(index = enums.IpAddressIndex.IP1)
```

Select/get the target IP address for message monitoring (method RsCmwEvdoSig.Configure.Mmonitor.enable) . The IP addresses are centrally managed from the 'Setup' dialog.

param index IP1 | IP2 | IP3

7.1.14 Application

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:APPLication:DSIGnaling
CONFigure:EVDO:SIGNaling<Instance>:APPLication:MODE
CONFigure:EVDO:SIGNaling<Instance>:APPLication
```

class Application

Application commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_dsignaling() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:APPLication:DSIGnaling
value: int = driver.configure.application.get_dsignaling()
```

Queries the stream and the state of the default signaling application. The response is fixed: The default signaling application is always enabled and assigned to stream 0.

return stream: Range: 0

get_mode() → RsCmwEvdoSig.enums.ApplicationMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:APPLication:MODE
value: enums.ApplicationMode = driver.configure.application.get_mode()
```

Selects the application to be instantiated.

return mode: FWD | REV | FAR | PACKet FWD: forward test application REV: reverse test application FAR: forward and reverse test application PACKet: packet application

get_value() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:APPLication
value: int = driver.configure.application.get_value()
```

Queries the stream occupied by the configured session application (see method RsCmwEvdoSig.Configure.Application.mode) . Setting this value has no effect as the selected stream is a result of the session negotiation.

return stream: Range: 1 to 3

set_mode(*mode: RsCmwEvdoSig.enums.ApplicationMode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:APPLication:MODE
driver.configure.application.set_mode(mode = enums.ApplicationMode.FAR)
```

Selects the application to be instantiated.

param mode FWD | REV | FAR | PACKet FWD: forward test application REV: reverse test application FAR: forward and reverse test application PACKet: packet application

set_value(*stream: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:APPLication
driver.configure.application.set_value(stream = 1)
```

Queries the stream occupied by the configured session application (see method RsCmwEvdoSig.Configure.Application.mode) . Setting this value has no effect as the selected stream is a result of the session negotiation.

param stream Range: 1 to 3

7.1.15 RfPower

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RFPower:EVDO
CONFigure:EVDO:SIGNaling<Instance>:RFPower:OUTPut
CONFigure:EVDO:SIGNaling<Instance>:RFPower:EPMode
CONFigure:EVDO:SIGNaling<Instance>:RFPower:MANual
CONFigure:EVDO:SIGNaling<Instance>:RFPower:EXPected
```

class RfPower

RfPower commands group definition. 7 total commands, 2 Sub-groups, 5 group commands

get_epmode() → RsCmwEvdoSig.enums.ExpPowerMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:EPMode
value: enums.ExpPowerMode = driver.configure.rfPower.get_epmode()
```

Selects the algorithm which the tester uses to configure its input path.

return exp_power_mode: MANual | OLRule | MAX | MIN | AUTO MANual: Manual setting, according to method RsCmwEvdoSig.Configure.RfPower.manual OLRule: According to open loop rule MAX: Maximum AT power MIN: Minimum AT power AUTO: Autoranging, according to received signal

get_evdo() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:EVDO
value: float = driver.configure.rfPower.get_evdo()
```

Defines the absolute power of the generated forward 1xEV-DO signal, excluding a possible AWGN contribution. The allowed value range can be calculated as follows: Range (EVDOPower) = Range (Output Power) - External Attenuation - AWGNPower Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet.

return evdo_power: Range: see above , Unit: dBm

get_expected() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:EXPected
value: float = driver.configure.rfPower.get_expected()
```

Queries the calculated value of the expected input power from the AT. The input power range is stated in the data sheet.

return exp_nom_power: Unit: dBm

get_manual() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:MANual
value: float = driver.configure.rfPower.get_manual()
```

Defines the expected absolute input power at the input connector for 'Expected Power Mode' = 'Manual' (method RsCmwEvdoSig.Configure.RfPower.epmode MANual) .

return manual_exp_power: Range: -47 dBm to 55 dBm, Unit: dBm

get_output() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:OUTPut
value: float = driver.configure.rfPower.get_output()
```

Queries the output power at the selected RF output connector: the sum of the '1xEV-DO Power' (method RsCmwEvdoSig. Configure.RfPower.evdo) and the AWGN (method RsCmwEvdoSig.Configure.RfPower.Level.awgn) . The allowed value: Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet.

return output_power: Range: see above , Unit: dBm

set_epmode(*exp_power_mode: RsCmwEvdoSig.enums.ExpPowerMode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:EPMode
driver.configure.rfPower.set_epmode(exp_power_mode = enums.ExpPowerMode.AUTO)
```

Selects the algorithm which the tester uses to configure its input path.

param exp_power_mode MANual | OLRule | MAX | MIN | AUTO MANual: Manual setting, according to method RsCmwEvdoSig.Configure.RfPower.manual OLRule: According to open loop rule MAX: Maximum AT power MIN: Minimum AT power AUTO: Autoranging, according to received signal

set_evdo(*evdo_power: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:EVDO
driver.configure.rfPower.set_evdo(evdo_power = 1.0)
```

Defines the absolute power of the generated forward 1xEV-DO signal, excluding a possible AWGN contribution. The allowed value range can be calculated as follows: Range (EVDOPower) = Range (Output Power) - External Attenuation - AWGNPower Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet.

param evdo_power Range: see above , Unit: dBm

set_manual(*manual_exp_power: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:MANual
driver.configure.rfPower.set_manual(manual_exp_power = 1.0)
```

Defines the expected absolute input power at the input connector for 'Expected Power Mode' = 'Manual' (method RsCmwEvdoSig.Configure.RfPower.epmode MANual) .

param manual_exp_power Range: -47 dBm to 55 dBm, Unit: dBm

Cloning the Group

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

Subgroups

7.1.15.1 Mode

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RFPower:MODE:AWGN
```

class Mode

Mode commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_awgn() → RsCmwEvdoSig.enums.AwgnMode

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:MODE:AWGN
value: enums.AwgnMode = driver.configure.rfPower.mode.get_awgn()
```

Selects the operating mode of the AWGN generator. The AWGN level range (method RsCmwEvdoSig.Configure.RfPower.Level.awgn) depends on the operating mode.

return awgn_mode: NORMal | HPOWer AWGN mode normal or high-power

set_awgn(*awgn_mode: RsCmwEvdoSig.enums.AwgnMode*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:MODE:AWGN
driver.configure.rfPower.mode.set_awgn(awgn_mode = enums.AwgnMode.HPOWer)
```

Selects the operating mode of the AWGN generator. The AWGN level range (method RsCmwEvdoSig.Configure.RfPower.Level.awgn) depends on the operating mode.

param awgn_mode NORMal | HPOWer AWGN mode normal or high-power

7.1.15.2 Level

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RFPower:LEVel:AWGN
```

class Level

Level commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_awgn() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:LEVel:AWGN
value: float or bool = driver.configure.rfPower.level.get_awgn()
```

Sets/gets the state and level of the AWGN generator relative to the '1xEV-DO Power' (method RsCmwEvdoSig.Configure. RfPower.evdo) . The AWGN level range depends on the operating mode of the AWGN generator (method RsCmwEvdoSig.Configure. RfPower.Mode.awgn) . With the 'set' command, the AWGN generator can be turned OFF or ON and the level can be set. If the level is set, the generator is automatically turned ON. The query returns either the OFF state or the level if the generator state is ON.

return awgn_level: Range: Between -25 dB and +4 dB (normal mode) or between -12 dB and 11.70 dB (high-power mode) , Unit: dB Additional OFF/ON disables/enables the AWGN signal

set_awgn(*awgn_level: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RFPower:LEVel:AWGN
driver.configure.rfPower.level.set_awgn(awgn_level = 1.0)
```

Sets/gets the state and level of the AWGN generator relative to the '1xEV-DO Power' (method RsCmwEvdoSig.Configure. RfPower.evdo) . The AWGN level range depends on the operating mode of the AWGN generator (method RsCmwEvdoSig.Configure. RfPower.Mode.awgn) . With the 'set' command, the AWGN generator can be turned OFF or ON and the level can be set. If the level is set, the generator is automatically turned ON. The query returns either the OFF state or the level if the generator state is ON.

param awgn_level Range: Between -25 dB and +4 dB (normal mode) or between -12 dB and 11.70 dB (high-power mode) , Unit: dB Additional OFF/ON disables/enables the AWGN signal

7.1.16 RpControl

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RPControl:PCBits
CONFigure:EVDO:SIGNaling<Instance>:RPControl:SSIZe
CONFigure:EVDO:SIGNaling<Instance>:RPControl:REPetition
CONFigure:EVDO:SIGNaling<Instance>:RPControl:RUN
```

class RpControl

RpControl commands group definition. 6 total commands, 1 Sub-groups, 4 group commands

get_pc_bits() → RsCmwEvdoSig.enums.PowerCtrlBits

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:PCBits
value: enums.PowerCtrlBits = driver.configure.rpControl.get_pc_bits()
```

Defines a power control bit pattern which the R&S CMW transmits to control the transmitter output power of the AT.

return pc_bits: AUTO | AUP | ADOWn | HOLD | PATTern AUTO: Active closed loop power control AUP: Power up bits ADOW: Power down bits HOLD: Alternating power up and power down bits PATT: Sends the user-specific segment bits executed by method RsCmwEvdoSig.Configure.RpControl.run.

get_repetition() → RsCmwEvdoSig.enums.Repeat

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:REPetition
value: enums.Repeat = driver.configure.rpControl.get_repetition()
```

Specifies the repetition mode of the pattern execution.

return repetition: SINGleshot | CONTinuous SINGleshot: the pattern execution is stopped after a single-shot CONTinuous: the pattern execution is repeated continuously and stopped by the method RsCmwEvdoSig.Configure.RpControl.run

get_run() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:RUN
value: bool = driver.configure.rpControl.get_run()
```

Starts and in continuous mode also stops the execution of the user-specific pattern.

return run_sequence_state: OFF | ON

get_ssize() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:SSIZe
value: float = driver.configure.rpControl.get_ssize()
```

Gets/sets the power control step size, i.e. the nominal change in mean output power per single power control bit.

return ssize: Range: 0.5 dB to 1 dB, Unit: dB

set_pc_bits(*pc_bits: RsCmwEvdoSig.enums.PowerCtrlBits*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:PCBits
driver.configure.rpControl.set_pc_bits(pc_bits = enums.PowerCtrlBits.ADOWn)
```

Defines a power control bit pattern which the R&S CMW transmits to control the transmitter output power of the AT.

param pc_bits AUTO | AUP | ADOWn | HOLD | PATTern AUTO: Active closed loop power control AUP: Power up bits ADOW: Power down bits HOLD: Alternating power up and power down bits PATT: Sends the user-specific segment bits executed by method RsCmwEvdoSig.Configure.RpControl.run.

set_repetition(*repetition: RsCmwEvdoSig.enums.Repeat*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:REPetition
driver.configure.rpControl.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the pattern execution.

param repetition SINGleshot | CONTinuous SINGleshot: the pattern execution is stopped after a single-shot CONTinuous: the pattern execution is repeated continuously and stopped by the method RsCmwEvdoSig.Configure.RpControl.run

set_run(*run_sequence_state: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:RUN
driver.configure.rpControl.set_run(run_sequence_state = False)
```

Starts and in continuous mode also stops the execution of the user-specific pattern.

param run_sequence_state OFF | ON

set_ssize(*ssize: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:SSIZe
driver.configure.rpControl.set_ssize(ssize = 1.0)
```

Gets/sets the power control step size, i.e. the nominal change in mean output power per single power control bit.

param ssize Range: 0.5 dB to 1 dB, Unit: dB

Cloning the Group

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

Subgroups

7.1.16.1 Segment

RepCap Settings

```
# Range: S1 .. S4
rc = driver.configure.rpControl.segment.repcap_segment_get()
driver.configure.rpControl.segment.repcap_segment_set(repcap.Segment.S1)
```

class Segment

Segment commands group definition. 2 total commands, 2 Sub-groups, 0 group commands Repeated Capability: Segment, default value after init: Segment.S1

Cloning the Group

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

Subgroups

7.1.16.1.1 Bits

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RPControl:SEGMent<Segment>:BITS
```

class Bits

Bits commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get(*segment=<Segment.Default: -1>*) → RsCmwEvdoSig.enums.SegmentBits

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:SEGMent<nr>:BITS
value: enums.SegmentBits = driver.configure.rpControl.segment.bits.get(segment␣
↪= repcap.Segment.Default)
```

Sets the user specific-power control bits.

param segment optional repeated capability selector. Default value: S1 (settable in the interface 'Segment')

return segment_bits: DOWN | UP | ALTernating All 0, all 1 or alternating

set(*segment_bits: RsCmwEvdoSig.enums.SegmentBits*, *segment=<Segment.Default: -1>*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:SEGMent<nr>:BITS
driver.configure.rpControl.segment.bits.set(segment_bits = enums.SegmentBits.
↪ALTernating, segment = repcap.Segment.Default)
```

Sets the user specific-power control bits.

param segment_bits DOWN | UP | ALTernating All 0, all 1 or alternating

param segment optional repeated capability selector. Default value: S1 (settable in the interface 'Segment')

7.1.16.1.2 Length

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RPControl:SEGMent<Segment>:LENGth
```

class Length

Length commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get(*segment=<Segment.Default: -1>*) → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:SEGMent<nr>:LENGth
value: int = driver.configure.rpControl.segment.length.get(segment = repcap.
↪Segment.Default)
```

Sets the length of the segment of the user-specific power control bits.

param segment optional repeated capability selector. Default value: S1 (settable in the interface 'Segment')

return segment_length: Segment length Range: 0 bits to 128 bits , Unit: bit

set(*segment_length: int*, *segment*=<*Segment.Default: -1*>) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RPControl:SEGMent<nr>:LENGth
driver.configure.rpControl.segment.length.set(segment_length = 1, segment =␣
↪repcap.Segment.Default)
```

Sets the length of the segment of the user-specific power control bits.

param segment_length Segment length Range: 0 bits to 128 bits , Unit: bit

param segment optional repeated capability selector. Default value: S1 (settable in the interface 'Segment')

7.1.17 System

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:TSOurce
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:DATE
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:TIME
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:SYNC
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:ATIMe
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:LSEConds
```

class System

System commands group definition. 8 total commands, 1 Sub-groups, 6 group commands

class DateStruct

Structure for reading output parameters. Fields:

- Day: int: Range: 1 to 31
- Month: int: Range: 1 to 12
- Year: int: Range: 2011 to 9999

class TimeStruct

Structure for reading output parameters. Fields:

- Hour: int: Range: 0 to 23
- Minute: int: Range: 0 to 59
- Second: int: Range: 0 to 59

get_atime() → RsCmwEvdoSig.enums.ApplyTimeAt

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:ATIMe
value: enums.ApplyTimeAt = driver.configure.system.get_atime()
```

Defines when the configured time source (method RsCmwEvdoSig.Configure.System.tsource) is applied to the SUU hosting the signaling application. Note that this setting is performance critical because applying the time at signal ON takes 3 to 4 seconds.

return apply_time_at: SUSO | EVER | NEXT SUSO (signaling unit startup only) : the time setting is only applied when the SUU starts up EVER: the time setting is applied at every signal ON NEXT: the time setting is applied at next signal ON; note that after the next signal ON the R&S CMW switches back to SUSO

get_date() → DateStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:DATE
value: DateStruct = driver.configure.system.get_date()
```

Date setting for CDMA system time source DATE (see method RsCmwEvdoSig.Configure.System.tsource) .

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

get_lseconds() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:LSEConds
value: int = driver.configure.system.get_lseconds()
```

Adjusts track of leap second correction to UTC.

return leap_seconds: Correction to the solar time Range: 0 to 255, Unit: s

get_sync() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:SYNC
value: str = driver.configure.system.get_sync()
```

Sets/queries the sync code. The sync code is required to synchronize the system time for 'Hybrid Mode on Two (or More) SUU': query the sync code generated by the 'synchronization master' (after SUU and set it on the 'synchronization slave'.

return sync_code: No help available

get_time() → TimeStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:TIME
value: TimeStruct = driver.configure.system.get_time()
```

Time setting for CDMA system time source 'Date / Time' (see method RsCmwEvdoSig.Configure.System.tsource) .

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

get_tsource() → RsCmwEvdoSig.enums.TimeSource

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:TSOurce
value: enums.TimeSource = driver.configure.system.get_tsource()
```

Queries/sets the time source for the derivation of the CMDA system time.

return source_time: CMWTime | DATE | SYNC CMWTime: CMW time (Windows time) DATE: Date and time as specified in method RsCmwEv-

doSig.Configure.System.date and method RsCmwEvdoSig.Configure.System.time SYNC: Sync code

set_atime(*apply_time_at: RsCmwEvdoSig.enums.ApplyTimeAt*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:ATIMe
driver.configure.system.set_atime(apply_time_at = enums.ApplyTimeAt.EVER)
```

Defines when the configured time source (method RsCmwEvdoSig.Configure.System.tsource) is applied to the SUU hosting the signaling application. Note that this setting is performance critical because applying the time at signal ON takes 3 to 4 seconds.

param apply_time_at SUSO | EVER | NEXT SUSO (signaling unit startup only) : the time setting is only applied when the SUU starts up EVER: the time setting is applied at every signal ON NEXT: the time setting is applied at next signal ON; note that after the next signal ON the R&S CMW switches back to SUSO

set_date(*value: RsCmwEvdoSig.Implementations.Configure_.System.System.DateStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:DATE
driver.configure.system.set_date(value = DateStruct())
```

Date setting for CDMA system time source DATE (see method RsCmwEvdoSig.Configure.System.tsource) .

param value see the help for DateStruct structure arguments.

set_lseconds(*leap_seconds: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:LSEConds
driver.configure.system.set_lseconds(leap_seconds = 1)
```

Adjusts track of leap second correction to UTC.

param leap_seconds Correction to the solar time Range: 0 to 255, Unit: s

set_sync(*sync_code: str*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:SYNC
driver.configure.system.set_sync(sync_code = r1)
```

Sets/queries the sync code. The sync code is required to synchronize the system time for 'Hybrid Mode on Two (or More) SUU': query the sync code generated by the 'synchronization master' (after SUU and set it on the 'synchronization slave'.

param sync_code No help available

set_time(*value: RsCmwEvdoSig.Implementations.Configure_.System.System.TimeStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:TIME
driver.configure.system.set_time(value = TimeStruct())
```

Time setting for CDMA system time source 'Date / Time' (see method RsCmwEvdoSig.Configure.System.tsource) .

param value see the help for TimeStruct structure arguments.

set_tsource(*source_time: RsCmwEvdoSig.enums.TimeSource*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:TSOurce
driver.configure.system.set_tsource(source_time = enums.TimeSource.CMWTime)
```

Queries/sets the time source for the derivation of the CMDA system time.

param source_time CMWTime | DATE | SYNC CMWTime: CMW time (Windows time) DATE: Date and time as specified in method RsCmwEvdoSig.Configure.System.date and method RsCmwEvdoSig.Configure.System.time SYNC: Sync code

Cloning the Group

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

Subgroups

7.1.17.1 LtOffset

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:LTOFfset:HEX
CONFigure:EVDO:SIGNaling<Instance>:SYSTem:LTOFfset
```

class LtOffset

LtOffset commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Sign: enums.SlopeType: NEGative | POSitive Position related to meridian NEGative: west from meridian POSitive: east from meridian
- Hour: int: Difference from UTC Range: 00 to 17, Unit: hour
- Minute: int: Difference from UTC Range: 00 to 59, Unit: min

get_hex() → str

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:LTOFfset:HEX
value: str = driver.configure.system.ltOffset.get_hex()
```

Displays time offset from UTC in hexadecimal format according to the local time zone.

return local_time_off_hex: LocalTimeOffset = (sign(h) *(abs(h) *60+m)) AND ((1UL11) -1) Range: #H000 to #HFFF

get_value() → ValueStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:LTOFfset
value: ValueStruct = driver.configure.system.ltOffset.get_value()
```

Defines the time offset from UTC according to the local time zone. Possible range is from -17:04 to +17:03

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

set_value(*value: RsCmwEvdoSig.Implementations.Configure_.System_.LtOffset.LtOffset.ValueStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:SYSTem:LTOFfset
driver.configure.system.ltOffset.set_value(value = ValueStruct())
```

Defines the time offset from UTC according to the local time zone. Possible range is from -17:04 to +17:03

param value see the help for ValueStruct structure arguments.

7.1.18 Ncell

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:RLMeutra
CONFigure:EVDO:SIGNaling<Instance>:NCELl:THRServing
CONFigure:EVDO:SIGNaling<Instance>:NCELl:MRTimer
```

class Ncell

Ncell commands group definition. 15 total commands, 4 Sub-groups, 3 group commands

get_mr_timer() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:MRTimer
value: int = driver.configure.ncell.get_mr_timer()
```

Maximum time for the access terminal to execute the cell reselection.

return max_reselection_timer: No help available

get_rlm_eutra() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:RLMeutra
value: int = driver.configure.ncell.get_rlm_eutra()
```

Configures the low reselection threshold value for LTE neighbor cells.

return rx_lev_min_eutra_common: No help available

get_thr_serving() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:THRServing
value: int = driver.configure.ncell.get_thr_serving()
```

Specifies the limit for pilot power level below which the AT triggers cell reselection to a neighbor cell in the candidate set.

return thresh_serving: Range: 0 to 63

set_mr_timer(*max_reselection_timer: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:MRTimer
driver.configure.ncell.set_mr_timer(max_reselection_timer = 1)
```

Maximum time for the access terminal to execute the cell reselection.

param max_reselection_timer Range: 0 to 15

set_rlm_eutra(*rx_lev_min_eutra_common: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:RLMeutra
driver.configure.ncell.set_rlm_eutra(rx_lev_min_eutra_common = 1)
```

Configures the low reselection threshold value for LTE neighbor cells.

param rx_lev_min_eutra_common Range: 0 to 96

set_thr_serving(*thresh_serving: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:THRServing
driver.configure.ncell.set_thr_serving(thresh_serving = 1)
```

Specifies the limit for pilot power level below which the AT triggers cell reselection to a neighbor cell in the candidate set.

param thresh_serving Range: 0 to 63

Cloning the Group

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

Subgroups

7.1.18.1 All

class All

All commands group definition. 2 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.18.1.1 Thresholds

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:ALL:THResholds:LOW
CONFigure:EVDO:SIGNaling<Instance>:NCELl:ALL:THResholds
```

class Thresholds

Thresholds commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class LowStruct

Structure for reading output parameters. Fields:

- Valid: bool: No parameter help available
- Low: int: No parameter help available

class ValueStruct

Structure for reading output parameters. Fields:

- Valid: bool: No parameter help available
- High: int: No parameter help available
- Low: int: No parameter help available

get_low() → LowStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:ALL:THResholds:LOW
value: LowStruct = driver.configure.ncell.all.thresholds.get_low()
```

No command help available

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

get_value() → ValueStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:ALL:THResholds
value: ValueStruct = driver.configure.ncell.all.thresholds.get_value()
```

No command help available

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

set_low(*value: RsCmwEvdoSig.Implementations.Configure_.Ncell_.All_.Thresholds.Thresholds.LowStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:ALL:THResholds:LOW
driver.configure.ncell.all.thresholds.set_low(value = LowStruct())
```

No command help available

param value see the help for LowStruct structure arguments.

set_value(*value:*
RsCmwEvdoSig.Implementations.Configure_.Ncell_.All_.Thresholds.Thresholds.ValueStruct) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:ALL:THResholds
driver.configure.ncell.all.thresholds.set_value(value = ValueStruct())
```

No command help available

param value see the help for ValueStruct structure arguments.

7.1.18.2 Evdo

class Evdo

Evdo commands group definition. 3 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.18.2.1 Cell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.configure.ncell.evdo.cell.repcap_cellNo_get()
driver.configure.ncell.evdo.cell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:EVDO:CELL<CellNo>
```

class Cell

Cell commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: CellNo, default value after init: CellNo.Nr1

class CellStruct

Structure for setting input parameters. Fields:

- Enable: bool: No parameter help available
- Band_Class: enums.BandClass: No parameter help available
- Channel: int: No parameter help available
- Cell_Id: int: No parameter help available

get(*cellNo=<CellNo.Default: -1>*) → CellStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:EVDO:CELL<n>
value: CellStruct = driver.configure.ncell.evdo.cell.get(cellNo = repcap.CellNo.
↪Default)
```

No command help available

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

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

set(*structure: RsCmwEvdoSig.Implementations.Configure_.Ncell_.Evdo_.Cell.Cell.CellStruct*, *cellNo=<CellNo.Default: -1>*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:EVDO:CELL<n>
driver.configure.ncell.evdo.cell.set(value = [PROPERTY_STRUCT_NAME](), cellNo =␣
↪repcap.CellNo.Default)
```

No command help available

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

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

Cloning the Group

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

7.1.18.2.2 Thresholds

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:EVDO:THResholds:LOW
CONFigure:EVDO:SIGNaling<Instance>:NCELl:EVDO:THResholds
```

class Thresholds

Thresholds commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

get_low() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:EVDO:THResholds:LOW
value: int = driver.configure.ncell.evdo.thresholds.get_low()
```

No command help available

return low: No help available

get_value() → ValueStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:EVDO:THResholds
value: ValueStruct = driver.configure.ncell.evdo.thresholds.get_value()
```

No command help available

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

set_low(*low: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:EVDO:THResholds:LOW
driver.configure.ncell.evdo.thresholds.set_low(low = 1)
```

No command help available

param low No help available

set_value(*value: RsCmwEvdoSig.Implementations.Configure_.Ncell_.Evdo_.Thresholds.Thresholds.ValueStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:EVDO:THResholds
driver.configure.ncell.evdo.thresholds.set_value(value = ValueStruct())
```

No command help available

param value see the help for ValueStruct structure arguments.

7.1.18.3 Cdma

class Cdma

Cdma commands group definition. 3 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.18.3.1 Cell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.configure.ncell.cdma.cell.repcap_cellNo_get()
driver.configure.ncell.cdma.cell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:CDMA:CELL<CellNo>
```

class Cell

Cell commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: CellNo, default value after init: CellNo.Nr1

class CellStruct

Structure for setting input parameters. Fields:

- Enable: bool: No parameter help available
- Band_Class: enums.BandClass: No parameter help available
- Channel: int: No parameter help available
- Cell_Id: int: No parameter help available

get(*cellNo*=<*CellNo.Default: -1*>) → CellStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:CDMA:CELL<n>
value: CellStruct = driver.configure.ncell.cdma.cell.get(cellNo = repcap.CellNo.
↪Default)
```

No command help available

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

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

set(*structure: RsCmwEvdoSig.Implementations.Configure_.Ncell_.Cdma_.Cell.Cell.CellStruct*, *cellNo*=<*CellNo.Default: -1*>) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:CDMA:CELL<n>
driver.configure.ncell.cdma.cell.set(value = [PROPERTY_STRUCT_NAME](), cellNo =␣
↪repcap.CellNo.Default)
```

No command help available

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

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

Cloning the Group

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

7.1.18.3.2 Thresholds

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:CDMA:THResholds:LOW
CONFigure:EVDO:SIGNaling<Instance>:NCELl:CDMA:THResholds
```

class Thresholds

Thresholds commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

get_low() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:CDMA:THResholds:LOW
value: int = driver.configure.ncell.cdma.thresholds.get_low()
```

No command help available

return low: No help available

get_value() → ValueStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:CDMA:THResholds
value: ValueStruct = driver.configure.ncell.cdma.thresholds.get_value()
```

No command help available

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

set_low(*low: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:CDMA:THResholds:LOW
driver.configure.ncell.cdma.thresholds.set_low(low = 1)
```

No command help available

param low No help available

set_value(*value: RsCmwEvdoSig.Implementations.Configure_.Ncell_.Cdma_.Thresholds.Thresholds.ValueStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:CDMA:THResholds
driver.configure.ncell.cdma.thresholds.set_value(value = ValueStruct())
```

No command help available

param value see the help for ValueStruct structure arguments.

7.1.18.4 Lte

class Lte
Lte commands group definition. 4 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.18.4.1 Cell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.configure.ncell.lte.cell.repcap_cellNo_get()
driver.configure.ncell.lte.cell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:LTE:CELL<CellNo>
```

class Cell
Cell commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: CellNo, default value after init: CellNo.Nr1

class CellStruct
Structure for setting input parameters. Fields:

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.LteBand: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16 | OB17 | OB18 | OB19 | OB20 | OB21 | OB22 | OB23 | OB24 | OB25 | OB26 | OB27 | OB28 | OB29 | OB30 | OB31 | OB32 | OB33 | OB34 | OB35 | OB36 | OB37 | OB38 | OB39 | OB40 | OB41 | OB42 | OB43 | OB44 | UDEFined OB1, ..., OB44: operating band 1 to 44 UDEFined: user-defined band
- Channel: int: Downlink channel number Range: depends on operating band, see tables below

get(*cellNo=<CellNo.Default: -1>*) → CellStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:CELL<n>
value: CellStruct = driver.configure.ncell.lte.cell.get(cellNo = repcap.CellNo.
↪Default)
```

Configures an entry of the neighbor cell list for LTE.

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

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

set(*structure: RsCmwEvdoSig.Implementations.Configure_.Ncell_.Lte_.Cell.Cell.CellStruct*, *cellNo=<CellNo.Default: -1>*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:CELL<n>
driver.configure.ncell.lte.cell.set(value = [PROPERTY_STRUCT_NAME](), cellNo =␣
↪repcap.CellNo.Default)
```

Configures an entry of the neighbor cell list for LTE.

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

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

Cloning the Group

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

7.1.18.4.2 Thresholds

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:LTE:THResholds:LOW
CONFigure:EVDO:SIGNaling<Instance>:NCELl:LTE:THResholds
```

class Thresholds

Thresholds commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

get_low() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:THResholds:LOW
value: int = driver.configure.ncell.lte.thresholds.get_low()
```

No command help available

return low: No help available

get_value() → ValueStruct

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:THResholds
value: ValueStruct = driver.configure.ncell.lte.thresholds.get_value()
```

No command help available

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

set_low(*low: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:THResholds:LOW
driver.configure.ncell.lte.thresholds.set_low(low = 1)
```

No command help available

param low No help available

set_value(*value: RsCmwEvdoSig.Implementations.Configure_.Ncell_.Lte_.Thresholds.Thresholds.ValueStruct*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:THResholds
driver.configure.ncell.lte.thresholds.set_value(value = ValueStruct())
```

No command help available

param value see the help for ValueStruct structure arguments.

7.1.18.4.3 Thrx<NeighborCell>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.configure.ncell.lte.thrx.repcap_neighborCell_get()
driver.configure.ncell.lte.thrx.repcap_neighborCell_set(repcap.NeighborCell.Nr1)
```

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:NCELl:LTE:THRX<NeighborCell>
```

class Thrx

Thrx commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: NeighborCell, default value after init: NeighborCell.Nr1

get(*neighborCell=<NeighborCell.Default: -1>*) → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:THRX<n>
value: int = driver.configure.ncell.lte.thrx.get(neighborCell = repcap.
↪NeighborCell.Default)
```

Specifies the minimum required quality threshold of the reselection target cell.

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

return lte_thresh_x: Range: 0 to 31

set(*lte_thresh_x: int, neighborCell=<NeighborCell.Default: -1>*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:NCELl:LTE:THRX<n>
driver.configure.ncell.lte.thrx.set(lte_thresh_x = 1, neighborCell = repcap.
↪NeighborCell.Default)
```

Specifies the minimum required quality threshold of the reselection target cell.

param lte_thresh_x Range: 0 to 31

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

Cloning the Group

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

7.1.19 Connection

class Connection

Connection commands group definition. 3 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.19.1 Edau

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:CONNection:EDAU:ENABle
CONFigure:EVDO:SIGNaling<Instance>:CONNection:EDAU:NSEGment
CONFigure:EVDO:SIGNaling<Instance>:CONNection:EDAU:NID
```

class Edau

Edau commands group definition. 3 total commands, 0 Sub-groups, 3 group commands

get_enable() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CONNection:EDAU:ENABle
value: bool = driver.configure.connection.edau.get_enable()
```

Enables use of an external DAU.

return enable: ON | OFF

get_nid() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CONNection:EDAU:NID
value: int = driver.configure.connection.edau.get_nid()
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

return idn: Range: 1 to 254

get_nsegment() → RsCmwEvdoSig.enums.NetworkSegment

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CONNection:EDAU:NSEGment
value: enums.NetworkSegment = driver.configure.connection.edau.get_nsegment()
```

Specifies the network segment of the instrument where the external DAU is installed.

return network_segment: A | B | C

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CONNection:EDAU:ENABle
driver.configure.connection.edau.set_enable(enable = False)
```

Enables use of an external DAU.

param enable ON | OFF

set_nid(*idn: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CONNection:EDAU:NID
driver.configure.connection.edau.set_nid(idn = 1)
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

param idn Range: 1 to 254

set_nsegment(*network_segment: RsCmwEvdoSig.enums.NetworkSegment*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:CONNection:EDAU:NSEGment
driver.configure.connection.edau.set_nsegment(network_segment = enums.
↪NetworkSegment.A)
```

Specifies the network segment of the instrument where the external DAU is installed.

param network_segment A | B | C

7.1.20 RxQuality

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:UPERiod
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:REPetition
```

class RxQuality

RxQuality commands group definition. 24 total commands, 10 Sub-groups, 2 group commands

get_repetition() → RsCmwEvdoSig.enums.Repeat

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:REPetition
value: enums.Repeat = driver.configure.rxQuality.get_repetition()
```

No command help available

return repetition: No help available

get_uperiod() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:UPERiod
value: float = driver.configure.rxQuality.get_uperiod()
```

Defines the time interval after which the R&S CMW evaluates and displays a new set of measurement results.

return update_period: Range: 0.25 s to 2 s, Unit: s

set_repetition(*repetition: RsCmwEvdoSig.enums.Repeat*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:REPetition
driver.configure.rxQuality.set_repetition(repetition = enums.Repeat.CONTinuous)
```

No command help available

param repetition No help available

set_uperiod(*update_period: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:UPERiod
driver.configure.rxQuality.set_uperiod(update_period = 1.0)
```

Defines the time interval after which the R&S CMW evaluates and displays a new set of measurement results.

param update_period Range: 0.25 s to 2 s, Unit: s

Cloning the Group

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

Subgroups

7.1.20.1 Carrier

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:CARRier:SELect
```

class Carrier

Carrier commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_select() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:CARRier:SELect
value: int = driver.configure.rxQuality.carrier.get_select()
```

Gets/sets the carrier whose results are retrieved in subsequent FETch/READ/CALCulate commands. Applies to PL subtype 3 only.

return selected_carrier: Range: 0 to 2

set_select(*selected_carrier: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:CARRier:SELect
driver.configure.rxQuality.carrier.set_select(selected_carrier = 1)
```

Gets/sets the carrier whose results are retrieved in subsequent FETch/READ/CALCulate commands. Applies to PL subtype 3 only.

param selected_carrier Range: 0 to 2

7.1.20.2 Rstatistics

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RSTatistics
```

class Rstatistics

Rstatistics commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

set() → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RSTatistics
driver.configure.rxQuality.rstatistics.set()
```

Clears the statistics for all receiver quality measurements and restarts the measurements.

set_with_opc() → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RSTatistics
driver.configure.rxQuality.rstatistics.set_with_opc()
```

Clears the statistics for all receiver quality measurements and restarts the measurements.

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

7.1.20.3 Per

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:PER:TOUT
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:PER:REPetition
```

class Per

Per commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_repetition() → RsCmwEvdoSig.enums.Repeat

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:PER:REPetition
value: enums.Repeat = driver.configure.rxQuality.per.get_repetition()
```

Specifies the repetition mode of the packet error rate (PER) measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwEvdoSig.Configure.RxQuality.FlPer. mtpsent or method RsCmwEvdoSig.Configure.RxQuality.RlPer.mpSent to determine the number of test packets per single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:PER:TOUT
value: float = driver.configure.rxQuality.per.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 ([ON | OFF] key or [RESTART | STOP] key) . 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: Unit: s

set_repetition(*repetition: RsCmwEvdoSig.enums.Repeat*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:PER:REPetition
driver.configure.rxQuality.per.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Specifies the repetition mode of the packet error rate (PER) measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continu-

ously. Use method RsCmwEvdoSig.Configure.RxQuality.FlPer. mtpsent or method RsCmwEvdoSig.Configure.RxQuality.RlPer.mpSent to determine the number of test packets per single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:PER:TOUT
driver.configure.rxQuality.per.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 ([ON | OFF] key or [RESTART | STOP] key) . 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 Unit: s

7.1.20.4 FlPer

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:FLPer:MTPSent
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:FLPer:SCONdition
```

class FlPer

FlPer commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_mtpsent() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:FLPer:MTPSent
value: int = driver.configure.rxQuality.flPer.get_mtpsent()
```

Defines the length of a single shot forward link PER measurement, i.e. the maximum number of test packets sent.

return max_test_pack_sent: Range: 1 to 10000

get_scondition() → RsCmwEvdoSig.enums.PerStopCondition

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:FLPer:SCONdition
value: enums.PerStopCondition = driver.configure.rxQuality.flPer.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 | ALEXceeded | MCLexceeded | MPERexceeded NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit

is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MPERexceeded: Stop if max. PER is exceeded

set_mtpsent(*max_test_pack_sent: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:FLPer:MTPSent
driver.configure.rxQuality.flPer.set_mtpsent(max_test_pack_sent = 1)
```

Defines the length of a single shot forward link PER measurement, i.e. the maximum number of test packets sent.

param max_test_pack_sent Range: 1 to 10000

set_scondition(*stop_condition: RsCmwEvdoSig.enums.PerStopCondition*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:FLPer:SCONdition
driver.configure.rxQuality.flPer.set_scondition(stop_condition = enums.
↪PerStopCondition.ALEXceeded)
```

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 | ALEXceeded | MCLexceeded | MPERexceeded NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MPERexceeded: Stop if max. PER is exceeded

7.1.20.5 RlPer

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RLPer:MPSent
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RLPer:SCONdition
```

class RlPer

RlPer commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_mp_sent() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RLPer:MPSent
value: int = driver.configure.rxQuality.rlPer.get_mp_sent()
```

Defines the length of a single shot reverse link PER measurement, i.e. the maximum number of test packets sent by the AT.

return max_packets_sent: Range: 1 to 10000

get_scondition() → RsCmwEvdoSig.enums.PerStopCondition

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RLPer:SCONdition
value: enums.PerStopCondition = driver.configure.rxQuality.rlPer.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 | ALEXceeded | MCLexceeded | MPERexceeded NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MPERexceeded: Stop if max. PER is exceeded

set_mp_sent(*max_packets_sent: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RLPer:MPSent
driver.configure.rxQuality.rlPer.set_mp_sent(max_packets_sent = 1)
```

Defines the length of a single shot reverse link PER measurement, i.e. the maximum number of test packets sent by the AT.

param max_packets_sent Range: 1 to 10000

set_scondition(*stop_condition: RsCmwEvdoSig.enums.PerStopCondition*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RLPer:SCONdition
driver.configure.rxQuality.rlPer.set_scondition(stop_condition = enums.
↪PerStopCondition.ALEXceeded)
```

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 | ALEXceeded | MCLexceeded | MPERexceeded NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MPERexceeded: Stop if max. PER is exceeded

7.1.20.6 Throughput

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:THRoughput:TOUT
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:THRoughput:REPetition
```

class Throughput

Throughput commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_repetition() → RsCmwEvdoSig.enums.Repeat

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:THRoughput:REPetition
value: enums.Repeat = driver.configure.rxQuality.throughput.get_repetition()
```

Specifies the repetition mode of the performance (throughput) measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwEvdoSig.Configure.RxQuality. FlPerformance.mframes or method RsCmwEvdoSig.Configure.RxQuality.RlPerformance.mframes to determine the number of test packets per single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:THRoughput:TOUT
value: float = driver.configure.rxQuality.throughput.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 ([ON | OFF] key or [RESTART | STOP] key) . 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: Unit: s

set_repetition(*repetition: RsCmwEvdoSig.enums.Repeat*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:THRoughput:REPetition
driver.configure.rxQuality.throughput.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Specifies the repetition mode of the performance (throughput) measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwEvdoSig.Configure.RxQuality. FlPerformance.mframes or method RsCmwEvdoSig.Configure.RxQuality.RlPerformance.mframes to determine the number of test packets per single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:THRoughput:TOUT
driver.configure.rxQuality.throughput.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 ([ON | OFF] key or [RESTART | STOP] key) . 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 Unit: s

7.1.20.7 FlPerformance

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:FLPFormance:MFRames
```

class FlPerformance

FlPerformance commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_mframes() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:FLPFormance:MFRames
value: int = driver.configure.rxQuality.flPerformance.get_mframes()
```

Defines the maximum duration of the 'Forward Link Throughput' / 'Reverse Link Throughput' measurement as a number of 26. 66 ms CDMA2000 frames.

return max_frames: Range: 1 to 10000

set_mframes(*max_frames: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:FLPFormance:MFRames
driver.configure.rxQuality.flPerformance.set_mframes(max_frames = 1)
```

Defines the maximum duration of the 'Forward Link Throughput' / 'Reverse Link Throughput' measurement as a number of 26. 66 ms CDMA2000 frames.

param max_frames Range: 1 to 10000

7.1.20.8 RlPerformance

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RLPFormance:MFRames
```

class RlPerformance

RlPerformance commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_mframes() → int

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RLPFormance:MFRames
value: int = driver.configure.rxQuality.rlPerformance.get_mframes()
```

Defines the maximum duration of the 'Forward Link Throughput' / 'Reverse Link Throughput' measurement as a number of 26. 66 ms CDMA2000 frames.

return max_frames: Range: 1 to 10000

set_mframes(*max_frames: int*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RLPFormance:MFRames
driver.configure.rxQuality.rlPerformance.set_mframes(max_frames = 1)
```

Defines the maximum duration of the 'Forward Link Throughput' / 'Reverse Link Throughput' measurement as a number of 26. 66 ms CDMA2000 frames.

param max_frames Range: 1 to 10000

7.1.20.9 Result

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RESult:FLPer
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RESult:RLPer
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RESult:FLPFormance
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RESult:RLPFormance
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:RESult:IPSTatistics
```

class Result

Result commands group definition. 5 total commands, 0 Sub-groups, 5 group commands

get_fl_per() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:FLPer
value: bool = driver.configure.rxQuality.result.get_fl_per()
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

return enable: OFF | ON

get_fl_performance() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:FLPFormance
value: bool = driver.configure.rxQuality.result.get_fl_performance()
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

return enable: OFF | ON

get_ip_statistics() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:IPSTatistics
value: bool = driver.configure.rxQuality.result.get_ip_statistics()
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

return enable: OFF | ON

get_rl_per() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:RLPer
value: bool = driver.configure.rxQuality.result.get_rl_per()
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

return enable: OFF | ON

get_rl_performance() → bool

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:RLPFormance
value: bool = driver.configure.rxQuality.result.get_rl_performance()
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

return enable: OFF | ON

set_fl_per(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:FLPer
driver.configure.rxQuality.result.set_fl_per(enable = False)
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

param enable OFF | ON

set_fl_performance(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:FLPFormance
driver.configure.rxQuality.result.set_fl_performance(enable = False)
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

param enable OFF | ON

set_ip_statistics(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:IPSTatistics
driver.configure.rxQuality.result.set_ip_statistics(enable = False)
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

param enable OFF | ON

set_rl_per(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:RLPer
driver.configure.rxQuality.result.set_rl_per(enable = False)
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

param enable OFF | ON

set_rl_performance(*enable: bool*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:RESult:RLPFormance
driver.configure.rxQuality.result.set_rl_performance(enable = False)
```

Enables/disables the view of the following RX measurements as indicated by the last mnemonics: 'Forward Link PER', 'Forward Link Throughput', 'Reverse Link PER', 'Reverse Link Throughput' and 'RLP & IP Statistics' (data) . For a disabled view, results are not displayed or calculated.

param enable OFF | ON

7.1.20.10 Limit

class Limit

Limit commands group definition. 5 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.20.10.1 Per

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:LIMit:PER:EVALuation
```

class Per

Per commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_evaluation() → RsCmwEvdoSig.enums.PerEvaluation

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:PER:EVALuation
value: enums.PerEvaluation = driver.configure.rxQuality.limit.per.get_
↪evaluation()
```

Defines whether the limits specified for the forward and reverse link packet error rate (PER) measurements is evaluated per carrier or over all carriers. This setting only affects multi-carrier evaluations.

return limit_evaluation: PERCarrier | ALLCarriers

set_evaluation(*limit_evaluation: RsCmwEvdoSig.enums.PerEvaluation*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:PER:EVALuation
driver.configure.rxQuality.limit.per.set_evaluation(limit_evaluation = enums.
↪PerEvaluation.ALLCarriers)
```

Defines whether the limits specified for the forward and reverse link packet error rate (PER) measurements is evaluated per carrier or over all carriers. This setting only affects multi-carrier evaluations.

param limit_evaluation PERCarrier | ALLCarriers

7.1.20.10.2 FlPer

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:LIMit:FLPer:MPER
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:LIMit:FLPer:CLEVel
```

class FlPer

FlPer commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_clevel() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:FLPer:CLEVel
value: float = driver.configure.rxQuality.limit.flPer.get_clevel()
```

Defines the minimum confidence level for the forward / reverse link PER measurement.

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

get_mper() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:FLPer:MPER
value: float = driver.configure.rxQuality.limit.flPer.get_mper()
```

Defines an upper limit for the measured forward / reverse link packet error ratio (PER) .

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

set_clevel(*min_confid_level: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:FLPer:CLEVel
driver.configure.rxQuality.limit.flPer.set_clevel(min_confid_level = 1.0)
```

Defines the minimum confidence level for the forward / reverse link PER measurement.

param min_confid_level Range: 0 % to 100 %, Unit: %

set_mper(*max_per: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:FLPer:MPER
driver.configure.rxQuality.limit.flPer.set_mper(max_per = 1.0)
```

Defines an upper limit for the measured forward / reverse link packet error ratio (PER) .

param max_per Range: 0 % to 100 %, Unit: %

7.1.20.10.3 RlPer

SCPI Commands

```
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:LIMit:RLPer:MPER
CONFigure:EVDO:SIGNaling<Instance>:RXQuality:LIMit:RLPer:CLEVel
```

class RlPer

RlPer commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_clevel() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:RLPer:CLEVel
value: float = driver.configure.rxQuality.limit.rlPer.get_clevel()
```

Defines the minimum confidence level for the forward / reverse link PER measurement.

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

get_mper() → float

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:RLPer:MPER
value: float = driver.configure.rxQuality.limit.rlPer.get_mper()
```

Defines an upper limit for the measured forward / reverse link packet error ratio (PER) .

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

set_clevel(*min_confid_level: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:RLPer:CLEVel
driver.configure.rxQuality.limit.rlPer.set_clevel(min_confid_level = 1.0)
```

Defines the minimum confidence level for the forward / reverse link PER measurement.

param min_confid_level Range: 0 % to 100 %, Unit: %

set_mper(*max_per: float*) → None

```
# SCPI: CONFigure:EVDO:SIGNaling<instance>:RXQuality:LIMit:RLPer:MPER
driver.configure.rxQuality.limit.rlPer.set_mper(max_per = 1.0)
```

Defines an upper limit for the measured forward / reverse link packet error ratio (PER) .

param max_per Range: 0 % to 100 %, Unit: %

7.2 Sense

class Sense
Sense commands group definition. 13 total commands, 6 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.1 Test

class Test
Test commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.1.1 Rx

class Rx
Rx commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.1.1.1 Power

SCPI Commands

```
SENSe:EVDO:SIGNaling<Instance>:TEST:RX:POWer:STATe
```

class Power
Power commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_state() → RsCmwEvdoSig.enums.RxSignalState

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:TEST:RX:POWer:STATe
value: enums.RxSignalState = driver.sense.test.rx.power.get_state()
```

Queries the quality of the RX signal from the connected AT.

return state: NAV | LOW | OK | HIGH NAV: no signal from AT detected LOW: the AT power is below the expected range OK: the AT power is in the expected range HIGH: the AT power is above the expected range

7.2.2 IqOut

class IqOut

IqOut commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.2.1 Path<Path>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.sense.iqOut.path.repcap_path_get()
driver.sense.iqOut.path.repcap_path_set(repcap.Path.Nr1)
```

SCPI Commands

```
SENSe:EVDO:SIGNaling<Instance>:IQOut:PATH<Path>
```

class Path

Path commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: Path, default value after init: Path.Nr1

class GetStruct

Response structure. Fields:

- Sample_Rate: enums.SampleRate: M100 Fixed value, indicating a sample rate of 100 Msps (100 MHz)
- Pep: float: Peak envelope power of the baseband signal Range: -60 dBFS to 0 dBFS , Unit: dBFS
- Crest_Factor: float: Crest factor of the baseband signal Range: 0 dB to 60 dB, Unit: dB

get(*path=<Path.Default: -1>*) → GetStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:IQOut:PATH<n>
value: GetStruct = driver.sense.iqOut.path.get(path = repcap.Path.Default)
```

Queries properties of the baseband signal at the I/Q output.

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

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

Cloning the Group

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

7.2.3 AnAddress

SCPI Commands

```
SENSe:EVDO:SIGNaling<Instance>:ANADdress:IPV<Const_IpV>
```

class AnAddress

AnAddress commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_ipv() → str

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:ANADdress:IPV<n>
value: str = driver.sense.anAddress.get_ipv()
```

No command help available

return an_address: No help available

7.2.4 AtAddress

class AtAddress

AtAddress commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.4.1 Ipv<IpAddress>

RepCap Settings

```
# Range: Version4 .. Version6
rc = driver.sense.atAddress.ipv.repcap_ipAddress_get()
driver.sense.atAddress.ipv.repcap_ipAddress_set(repcap.IpAddress.Version4)
```

SCPI Commands

```
SENSe:EVDO:SIGNaling<Instance>:ATADdress:IPV<IpAddress>
```

class Ipv

Ipv commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: IpAddress, default value after init: IpAddress.Version4

get(*ipAddress=<IpAddress.Default: -1>*) → str

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:ATADdress:IPV<n>
value: str = driver.sense.atAddress.ipv.get(ipAddress = repcap.IpAddress.
↪Default)
```

Returns the IPv4 address (<n> = 4) or the IPv6 prefix (<n> = 6) assigned to the AT by the DAU.

param ipAddress optional repeated capability selector. Default value: Version4 (settable in the interface 'Ipv')

return ip_address: IP address/prefix as string

Cloning the Group

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

7.2.5 RxQuality

class RxQuality

RxQuality commands group definition. 7 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.5.1 IpStatistics

SCPI Commands

```
SENSe:EVDO:SIGNaling<Instance>:RXQuality:IPSTatistics:STATe
SENSe:EVDO:SIGNaling<Instance>:RXQuality:IPSTatistics:RESet
SENSe:EVDO:SIGNaling<Instance>:RXQuality:IPSTatistics:RACK
SENSe:EVDO:SIGNaling<Instance>:RXQuality:IPSTatistics:NAK
SENSe:EVDO:SIGNaling<Instance>:RXQuality:IPSTatistics:SUMMary
SENSe:EVDO:SIGNaling<Instance>:RXQuality:IPSTatistics:PPPTotal
SENSe:EVDO:SIGNaling<Instance>:RXQuality:IPSTatistics:DRATe
```

class IpStatistics

IpStatistics commands group definition. 7 total commands, 0 Sub-groups, 7 group commands

class DrateStruct

Structure for reading output parameters. Fields:

- Rx: float: Data rate in receive direction Range: 0 kbit/s to 0.999999E+6 kbit/s
- Tx: float: Data rate in transmit direction Range: 0 kbit/s to 0.999999E+6 kbit/s

class NakStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of packets received in the last update period Range: 0 to 0.999999E+6
- Rx_Total: int: Total number of packets received since the beginning of the PPP connection Range: 0 to 0.999999E+6
- Tx: int: Number of packets transmitted in the last update period Range: 0 to 0.999999E+6
- Tx_Total: int: Total number of packets transmitted Range: 0 to 0.999999E+6

class PppTotalStruct

Structure for reading output parameters. Fields:

- Rx: int: Total size of data received Range: 0 KB to 0.999999E+6 KB
- Tx: int: Total size of data transmitted Range: 0 KB to 0.999999E+6 KB

class RackStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of packets received in the last update period Range: 0 to 0.999999E+6
- Rx_Total: int: Total number of packets received since the beginning of the PPP connection Range: 0 to 0.999999E+6
- Tx: int: Number of packets transmitted in the last update period Range: 0 to 0.999999E+6
- Tx_Total: int: Total number of packets transmitted Range: 0 to 0.999999E+6

class ResetStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of packets received in the last update period Range: 0 to 0.999999E+6
- Rx_Total: int: Total number of packets received since the beginning of the PPP connection Range: 0 to 0.999999E+6
- Tx: int: Number of packets transmitted in the last update period Range: 0 to 0.999999E+6
- Tx_Total: int: Total number of packets transmitted Range: 0 to 0.999999E+6

class SummaryStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of packets received in the last update period Range: 0 to 0.999999E+6
- Rx_Total: int: Total number of packets received since the beginning of the PPP connection Range: 0 to 0.999999E+6
- Tx: int: Number of packets transmitted in the last update period Range: 0 to 0.999999E+6
- Tx_Total: int: Total number of packets transmitted Range: 0 to 0.999999E+6

get_drate() → DrateStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:RXQuality:IPSTatistics:DRATe
value: DrateStruct = driver.sense.rxQuality.ipStatistics.get_drate()
```

Current received data rate in kbit/s, averaged over the update period.

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

get_nak() → NakStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:RXQuality:IPSTatistics:NAK
value: NakStruct = driver.sense.rxQuality.ipStatistics.get_nak()
```

Number of NAK control packets, requesting the retransmission of one or more data octets.

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

get_ppp_total() → PppTotalStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:RXQuality:IPSTatistics:PPPTotal
value: PppTotalStruct = driver.sense.rxQuality.ipStatistics.get_ppp_total()
```

Total number of bytes the R&S CMW received (Rx) and sent (Tx) since the beginning of the PPP connection.

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

get_rack() → RackStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:RXQuality:IPSTatistics:RACK
value: RackStruct = driver.sense.rxQuality.ipStatistics.get_rack()
```

Number of packets associated with RLP reset ACK messages, which are sent between AT and AN to complete the RLP reset procedure.

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

get_reset() → ResetStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:RXQuality:IPSTatistics:RESet
value: ResetStruct = driver.sense.rxQuality.ipStatistics.get_reset()
```

Number of packets associated with RLP reset messages, which are sent between AT and AN to reset RLP.

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

get_state() → str

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:RXQuality:IPSTatistics:STATe
value: str = driver.sense.rxQuality.ipStatistics.get_state()
```

Status of the RLP & IP statistics

return status: See table below

get_summary() → SummaryStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:RXQuality:IPSTatistics:SUMMary
value: SummaryStruct = driver.sense.rxQuality.ipStatistics.get_summary()
```

Total number of packets from the measured RLP messages. As the list contains all packet types, this value is equal to the total number of RLP packets received.

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

7.2.6 Elog

SCPI Commands

```
SENSe:EVDO:SIGNaling<Instance>:ELOG:LAST
SENSe:EVDO:SIGNaling<Instance>:ELOG:ALL
```

class Elog

Elog commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class AllStruct

Structure for reading output parameters. Fields:

- Timestamp: List[str]: Timestamp of the entry as string in the format 'hh:mm:ss'
- Category: List[enums.LogCategory]: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon
- Event: List[str]: Text string describing the event, e.g. 'Session Opened'

class LastStruct

Structure for reading output parameters. Fields:

- Timestamp: str: Timestamp of the entry as string in the format 'hh:mm:ss'
- Category: enums.LogCategory: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon

- Event: str: Text string describing the event, e.g. 'Session Opened'

get_all() → AllStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:ELOG:ALL
value: AllStruct = driver.sense.elog.get_all()
```

Queries all entries of the event log. For each entry, three parameters are returned, from oldest to latest entry: {<Timestamp>, <Category>, <Event>}entry 1, {<Timestamp>, <Category>, <Event>}entry 2, ...

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

get_last() → LastStruct

```
# SCPI: SENSe:EVDO:SIGNaling<instance>:ELOG:LAST
value: LastStruct = driver.sense.elog.get_last()
```

Queries the latest entry of the event log.

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

7.3 Route

SCPI Commands

```
ROUTe:EVDO:SIGNaling<Instance>
```

class Route

Route commands group definition. 9 total commands, 1 Sub-groups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SCEL | HMODe | HMLite | SCFading | HMFading SCEL: Standard cell HMOD: Hybrid mode HMLite: Hybrid mode lite SCFading: Standard cell fading HMFading: Hybrid mode with fading
- Controller: str: For future use - returned value not relevant
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path
- Iq_1_Connector: enums.TxConnector: DIG IQ OUT connector for the output path, only returned for scenarios with external fading

get_value() → ValueStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>
value: ValueStruct = driver.route.get_value()
```

Returns the configured routing settings. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

Cloning the Group

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

Subgroups

7.3.1 Scenario

SCPI Commands

```
ROUTe:EVDO:SIGNaling<Instance>:SCENario:SCELl
ROUTe:EVDO:SIGNaling<Instance>:SCENario:HMODe
ROUTe:EVDO:SIGNaling<Instance>:SCENario:HMLite
ROUTe:EVDO:SIGNaling<Instance>:SCENario
```

class Scenario

Scenario commands group definition. 8 total commands, 2 Sub-groups, 4 group commands

class HmliteStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

class HmodeStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

class ScellStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SCELl | HMODe | HMLite | SCFading | HMFading SCEL: Standard cell HMOD: Hybrid mode HMLite: Hybrid mode lite SCFading: Standard cell fading HMFading: Hybrid mode with fading
- Fader: enums.SourceInt: EXTernal | INTernal Only returned for fading scenario (SCF) Indicates whether internal or external fading is active.

get_hmlite() → HmliteStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMLite
value: HmliteStruct = driver.route.scenario.get_hmlite()
```

Activates the 'Hybrid Mode Lite' scenario and selects the signal path. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_hmode() → HmodeStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMODe
value: HmodeStruct = driver.route.scenario.get_hmode()
```

Activates the hybrid mode scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_scell() → ScellStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:SCELl
value: ScellStruct = driver.route.scenario.get_scell()
```

Activates the standalone scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_value() → ValueStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario
value: ValueStruct = driver.route.scenario.get_value()
```

Returns the active scenario.

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

set_hmlite(*value: RsCmwEvdoSig.Implementations.Route_.Scenario.Scenario.HmliteStruct*) → None

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMLite
driver.route.scenario.set_hmlite(value = HmliteStruct())
```

Activates the 'Hybrid Mode Lite' scenario and selects the signal path. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for HmliteStruct structure arguments.

set_hmode(*value: RsCmwEvdoSig.Implementations.Route_.Scenario.Scenario.HmodeStruct*) → None

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMODe
driver.route.scenario.set_hmode(value = HmodeStruct())
```

Activates the hybrid mode scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for HmodeStruct structure arguments.

set_scell(*value: RsCmwEvdoSig.Implementations.Route_.Scenario.Scenario.ScellStruct*) → None

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:SCELl
driver.route.scenario.set_scell(value = ScellStruct())
```

Activates the standalone scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ScellStruct structure arguments.

Cloning the Group

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

Subgroups

7.3.1.1 ScFading

SCPI Commands

```
ROUTe:EVDO:SIGNaling<Instance>:SCENario:SCFading:EXTernal
ROUTe:EVDO:SIGNaling<Instance>:SCENario:SCFading:INTernal
```

class ScFading

ScFading commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ExternalStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the output path

class InternalStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path

- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

get_external() → ExternalStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:SCFading[:EXTernal]
value: ExternalStruct = driver.route.scenario.scFading.get_external()
```

Activates the 'Standard Cell Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_internal() → InternalStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:SCFading:INTernal
value: InternalStruct = driver.route.scenario.scFading.get_internal()
```

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwEvdoSig.Implementations.Route_.Scenario_.ScFading.ScFading.ExternalStruct*) → None

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:SCFading[:EXTernal]
driver.route.scenario.scFading.set_external(value = ExternalStruct())
```

Activates the 'Standard Cell Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ExternalStruct structure arguments.

set_internal(*value: RsCmwEvdoSig.Implementations.Route_.Scenario_.ScFading.ScFading.InternalStruct*) → None

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:SCFading:INTernal
driver.route.scenario.scFading.set_internal(value = InternalStruct())
```

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.3.1.2 HmFading

SCPI Commands

```
ROUTe:EVDO:SIGNaling<Instance>:SCENario:HMFading:EXTernal
ROUTe:EVDO:SIGNaling<Instance>:SCENario:HMFading:INTernal
```

class HmFading

HmFading commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ExternalStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the output path

class InternalStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

get_external() → ExternalStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMFading[:EXTernal]
value: ExternalStruct = driver.route.scenario.hmFading.get_external()
```

Activates the 'Hybrid Mode Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_internal() → InternalStruct

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMFading:INTernal
value: InternalStruct = driver.route.scenario.hmFading.get_internal()
```

Activates the 'Hybrid Mode Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwEvdoSig.Implementations.Route_.Scenario_.HmFading.HmFading.ExternalStruct*) → None

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMFading[:EXTernal]
driver.route.scenario.hmFading.set_external(value = ExternalStruct())
```

Activates the 'Hybrid Mode Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ExternalStruct structure arguments.

set_internal(*value: RsCmwEvdoSig.Implementations.Route_.Scenario_.HmFading.HmFading.InternalStruct*) → None

```
# SCPI: ROUTe:EVDO:SIGNaling<instance>:SCENario:HMFading:INTernal
driver.route.scenario.hmFading.set_internal(value = InternalStruct())
```

Activates the 'Hybrid Mode Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.4 Source

class Source

Source commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.4.1 RfSettings

class RfSettings

RfSettings commands group definition. 2 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.4.1.1 Tx

SCPI Commands

```
SOURce:EVDO:SIGNaling<Instance>:RFSettings:TX:EATTenuation
```

class Tx

Tx commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_eattenuation() → float

```
# SCPI: SOURce:EVDO:SIGNaling<instance>:RFSettings:TX:EATTenuation
value: float = driver.source.rfSettings.tx.get_eattenuation()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the output connector.

return tx_ext_att: Range: -50 dB to 90 dB, Unit: dB

set_eattenuation(*tx_ext_att: float*) → None

```
# SCPI: SOURce:EVDO:SIGNaling<instance>:RFSettings:TX:EATTenuation
driver.source.rfSettings.tx.set_eattenuation(tx_ext_att = 1.0)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the output connector.

param tx_ext_att Range: -50 dB to 90 dB, Unit: dB

7.4.1.2 Rx

SCPI Commands

```
SOURce:EVDO:SIGNaling<Instance>:RFSettings:RX:EATTenuation
```

class Rx

Rx commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_eattenuation() → float

```
# SCPI: SOURce:EVDO:SIGNaling<instance>:RFSettings:RX:EATTenuation
value: float = driver.source.rfSettings.rx.get_eattenuation()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

return rx_ext_att: Range: -50 dB to 90 dB, Unit: dB

set_eattenuation(*rx_ext_att: float*) → None

```
# SCPI: SOURce:EVDO:SIGNaling<instance>:RFSettings:RX:EATTenuation
driver.source.rfSettings.rx.set_eattenuation(rx_ext_att = 1.0)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

param rx_ext_att Range: -50 dB to 90 dB, Unit: dB

7.4.2 State

SCPI Commands

```
SOURce:EVDO:SIGNaling<Instance>:STATe:ALL
SOURce:EVDO:SIGNaling<Instance>:STATe
```

class State

State commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class AllStruct

Structure for reading output parameters. Fields:

- Main_State: enums.MainGenState: OFF | ON | RFHandover ON: generator has been turned on OFF: generator switched off RFHandover: Ready for handover, i.e. the signaling application is ready to receive an inter-RAT handover from another signaling application (e.g. LTE) , see 'Inter-RAT Handover' for details.
- Sync_State: enums.SyncState: PENDing | ADJusted PENDing: the generator has been turned on (off) but the signal is not yet (still) available ADJusted: the physical output signal corresponds to the main generator state (signal off for main state OFF, signal on for main state ON)

get_all() → AllStruct

```
# SCPI: SOURce:EVDO:SIGNaling<instance>:STATe:ALL
value: AllStruct = driver.source.state.get_all()
```

Returns detailed information about the '1xEV-DO Signaling' generator state.

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

get_value() → bool

```
# SCPI: SOURce:EVDO:SIGNaling<instance>:STATe
value: bool = driver.source.state.get_value()
```

Turns the 1xEV-DO signaling generator (the cell) off or on.

return main_state: No help available

set_value(*main_state: bool*) → None

```
# SCPI: SOURce:EVDO:SIGNaling<instance>:STATe
driver.source.state.set_value(main_state = False)
```

Turns the 1xEV-DO signaling generator (the cell) off or on.

param main_state No help available

7.5 Call

class Call
Call commands group definition. 2 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.5.1 Cswitched

SCPI Commands

```
CALL:EVDO:SIGNaling<Instance>:CSWitched:ACTion
```

class Cswitched
Cswitched commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

set_action(*cs_action: RsCmwEvdoSig.enums.CswitchedAction*) → None

```
# SCPI: CALL:EVDO:SIGNaling<instance>:CSWitched:ACTion
driver.call.cswitched.set_action(cs_action = enums.CswitchedAction.CLOSe)
```

Controls the setup and release of an 1xEV-DO connection. The command initiates a transition between different connection states; to be queried via method RsCmwEvdoSig.Cswitched.State.fetch. For details, refer to 'Connection States'.

param cs_action CONNect | DISConnect | CLOSe | HANDoff Transition between connection states

7.5.2 Handoff

SCPI Commands

```
CALL:EVDO:SIGNaling<Instance>:HANDoff:STARt
```

class Handoff
Handoff commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

start() → None

```
# SCPI: CALL:EVDO:SIGNaling<instance>:HANDoff:STARt
driver.call.handoff.start()
```

Initiates a handoff to the previously configured destination cell. After the handoff, the destination cell settings replace the current cell settings.

start_with_opc() → None

```
# SCPI: CALL:EVDO:SIGNaling<instance>:HANDoff:STARt
driver.call.handoff.start_with_opc()
```

Initiates a handoff to the previously configured destination cell. After the handoff, the destination cell settings replace the current cell settings.

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

7.6 Cswitched

class Cswitched

Cswitched commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.6.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:CSWitched:STATe
```

class State

State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → RsCmwEvdoSig.enums.ConnectionState

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:CSWitched:STATe
value: enums.ConnectionState = driver.cswitched.state.fetch()
```

Returns the connection state of an 1xEV-DO connection. Use method RsCmwEvdoSig.Call.Cswitched.action to initiate a transition between different connection states. The connection state changes to ON when the signaling generator is started (method RsCmwEvdoSig.Source.State.value ON) . To make sure that a forward 1xEV-DO signal is available, query the sector state: method RsCmwEvdoSig.Source.State.all must return ON, ADJ.

return cs_state: OFF | ON | IDLE | SNEGotiation | SOPen | PAGing | CONNected Connection state; for details refer to 'Connection States'.

7.7 Pdata

class Pdata
Pdata commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.7.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:PDATa:STATe
```

class State
State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → RsCmwEvdoSig.enums.PdState

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:PDATa:STATe
value: enums.PdState = driver.pdata.state.fetch()
```

Returns the state of the packet data (PPP) connection.

return pd_state: OFF | ON | DORMant | CONNected

7.8 Per

SCPI Commands

```
INITiate:EVDO:SIGNaling<Instance>:PER
STOP:EVDO:SIGNaling<Instance>:PER
ABORt:EVDO:SIGNaling<Instance>:PER
```

class Per
Per commands group definition. 5 total commands, 1 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:EVDO:SIGNaling<instance>:PER
driver.per.abort()
```

These remote commands control the packet error rate measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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.

abort_with_opc() → None

```
# SCPI: ABORt:EVDO:SIGNaling<instance>:PER
driver.per.abort_with_opc()
```

These remote commands control the packet error rate measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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

initiate() → None

```
# SCPI: INITiate:EVDO:SIGNaling<instance>:PER
driver.per.initiate()
```

These remote commands control the packet error rate measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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.

initiate_with_opc() → None

```
# SCPI: INITiate:EVDO:SIGNaling<instance>:PER
driver.per.initiate_with_opc()
```

These remote commands control the packet error rate measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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

stop() → None

```
# SCPI: STOP:EVDO:SIGNaling<instance>:PER
driver.per.stop()
```

These remote commands control the packet error rate measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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() → None

```
# SCPI: STOP:EVDO:SIGNaling<instance>:PER
driver.per.stop_with_opc()
```

These remote commands control the packet error rate measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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 RsCmwEvdoSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.8.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:PER:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwEvdoSig.enums.ResourceState

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:PER:STATe
value: enums.ResourceState = driver.per.state.fetch()
```

Queries the main measurement state. Use FETCh:...:STATe:ALL? to query the measurement state including the substates. Use INITiate..., STOP..., ABORt... to change the measurement state.

return state: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

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

Subgroups

7.8.1.1 All

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:PER:STATe:ALL
```

class All

All commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: all necessary adjustments finished, measurement running ('adjusted') INV: not applicable because main_state: OFF or RDY ('invalid')
- Resource_State: enums.ResourceState: QUE | ACT | INV QUE: measurement without resources, no results available ('queued') ACT: resources allocated, acquisition of results in progress but not complete ('active') INV: not applicable because main_state: OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:PER:STATe:ALL
value: FetchStruct = driver.per.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

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

7.9 Throughput

SCPI Commands

```
INITiate:EVDO:SIGNaling<Instance>:THRoughput
STOP:EVDO:SIGNaling<Instance>:THRoughput
ABORt:EVDO:SIGNaling<Instance>:THRoughput
```

class Throughput

Throughput commands group definition. 5 total commands, 1 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:EVDO:SIGNaling<instance>:THRoughput
driver.throughput.abort()
```

These remote commands control the throughput (performance) measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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.

abort_with_opc() → None

```
# SCPI: ABORt:EVDO:SIGNaling<instance>:THRoughput
driver.throughput.abort_with_opc()
```

These remote commands control the throughput (performance) measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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

initiate() → None

```
# SCPI: INITiate:EVDO:SIGNaling<instance>:THRoughput
driver.throughput.initiate()
```

These remote commands control the throughput (performance) measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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.

initiate_with_opc() → None

```
# SCPI: INITiate:EVDO:SIGNaling<instance>:THRoughput
driver.throughput.initiate_with_opc()
```

These remote commands control the throughput (performance) measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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

stop() → None

```
# SCPI: STOP:EVDO:SIGNaling<instance>:THRoughput
driver.throughput.stop()
```

These remote commands control the throughput (performance) measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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() → None

```
# SCPI: STOP:EVDO:SIGNaling<instance>:THRoughput
driver.throughput.stop_with_opc()
```

These remote commands control the throughput (performance) measurement - either forward or reverse, depending on the current test application mode (see method RsCmwEvdoSig.Configure.Application.mode) .

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 RsCmwEvdoSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.9.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:THRoughput:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwEvdoSig.enums.ResourceState

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:THRoughput:STATe
value: enums.ResourceState = driver.throughput.state.fetch()
```

Queries the main measurement state. Use FETCh:...:STATe:ALL? to query the measurement state including the substates. Use INITiate..., STOP..., ABORt... to change the measurement state.

return state: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

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

Subgroups

7.9.1.1 All

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:THRoughput:STATe:ALL
```

class All

All commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: all necessary adjustments finished, measurement running ('adjusted') INV: not applicable because main_state: OFF or RDY ('invalid')
- Resource_State: enums.ResourceState: QUE | ACT | INV QUE: measurement without resources, no results available ('queued') ACT: resources allocated, acquisition of results in progress but not complete ('active') INV: not applicable because main_state: OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:THRoughput:STATe:ALL
value: FetchStruct = driver.throughput.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

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

7.10 RxQuality

SCPI Commands

```
INITiate:EVDO:SIGNaling<Instance>:RXQuality
STOP:EVDO:SIGNaling<Instance>:RXQuality
ABORt:EVDO:SIGNaling<Instance>:RXQuality
```

class RxQuality

RxQuality commands group definition. 25 total commands, 5 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:EVDO:SIGNaling<instance>:RXQuality
driver.rxQuality.abort()
```

No command help available

abort_with_opc() → None

```
# SCPI: ABORt:EVDO:SIGNaling<instance>:RXQuality
driver.rxQuality.abort_with_opc()
```

No command help available

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

initiate() → None

```
# SCPI: INITiate:EVDO:SIGNaling<instance>:RXQuality
driver.rxQuality.initiate()
```

No command help available

initiate_with_opc() → None

```
# SCPI: INITiate:EVDO:SIGNaling<instance>:RXQuality
driver.rxQuality.initiate_with_opc()
```

No command help available

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

stop() → None

```
# SCPI: STOP:EVDO:SIGNaling<instance>:RXQuality
driver.rxQuality.stop()
```

No command help available

stop_with_opc() → None

```
# SCPI: STOP:EVDO:SIGNaling<instance>:RXQuality
driver.rxQuality.stop_with_opc()
```

No command help available

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

Cloning the Group

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

Subgroups

7.10.1 FlPer

SCPI Commands

```
READ:EVDO:SIGNaling<Instance>:RXQuality:FLPer
FETCh:EVDO:SIGNaling<Instance>:RXQuality:FLPer
CALCulate:EVDO:SIGNaling<Instance>:RXQuality:FLPer
```

class FlPer

FlPer commands group definition. 5 total commands, 2 Sub-groups, 3 group commands

class CalculateStruct
Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Per: float: Current forward link packet error rate (for the selected carrier) Range: 0 % to 100 %, Unit: %
- Confidence_Level: float: Confidence level (for the selected carrier) Range: 0 % to 100 %, Unit: %
- Packet_Errors: float: Number of detected packet errors (for the selected carrier) Range: 0 to 100E+3
- Test_Packets_Sent: float: Packets sent (on the selected carrier) Range: 0 to 100E+3
- Total_Per: float: No parameter help available
- Total_Conf_Level: float: No parameter help available
- Total_Pack_Errors: float: No parameter help available
- Total_Test_Psent: float: No parameter help available

class ResultData
Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Per: float: Current forward link packet error rate (for the selected carrier) Range: 0 % to 100 %, Unit: %
- Confidence_Level: float: Confidence level (for the selected carrier) Range: 0 % to 100 %, Unit: %
- Packet_Errors: int: Number of detected packet errors (for the selected carrier) Range: 0 to 100E+3
- Test_Packets_Sent: int: Packets sent (on the selected carrier) Range: 0 to 100E+3
- Total_Per: float: No parameter help available
- Total_Conf_Level: float: No parameter help available
- Total_Pack_Errors: int: No parameter help available
- Total_Test_Psent: int: No parameter help available

calculate() → CalculateStruct

```
# SCPI: CALCulate:EVDO:SIGNaling<instance>:RXQuality:FLPer
value: CalculateStruct = driver.rxQuality.flPer.calculate()
```

Returns the results of the 'Forward Link PER' measurement (for the selected carrier) , see 'Forward Link Packet Error Rate Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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:EVDO:SIGNaling<instance>:RXQuality:FLPer
value: ResultData = driver.rxQuality.flPer.fetch()
```

Returns the results of the 'Forward Link PER' measurement (for the selected carrier) , see 'Forward Link Packet Error Rate Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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:EVDO:SIGNaling<instance>:RXQuality:FLPer
value: ResultData = driver.rxQuality.flPer.read()
```

Returns the results of the 'Forward Link PER' measurement (for the selected carrier) , see 'Forward Link Packet Error Rate Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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.

Cloning the Group

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

Subgroups

7.10.1.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:FLPer:STATe
```

class State

State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → str

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:FLPer:STATe
value: str = driver.rxQuality.flPer.state.fetch()
```

Returns a string containing status information about the measurement.

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

return status: See table below

7.10.1.2 Cstate

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:FLPer:CSTate
```

class Cstate

Cstate commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → List[RsCmwEvdoSig.enums.CarrierStatus]

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:FLPer:CSTate
value: List[enums.CarrierStatus] = driver.rxQuality.flPer.cstate.fetch()
```

Returns status information about the carrier.

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

return carrier_status: OK | VIOLated | STALe | INACtive

7.10.2 RlPer

SCPI Commands

```
READ:EVDO:SIGNaling<Instance>:RXQuality:RLPer
FETCh:EVDO:SIGNaling<Instance>:RXQuality:RLPer
CALCulate:EVDO:SIGNaling<Instance>:RXQuality:RLPer
```

class RlPer

RlPer commands group definition. 5 total commands, 2 Sub-groups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Macp_Kts_Transm: float: No parameter help available
- Rl_Macp_Kts_Errors: float: No parameter help available
- Rl_Conf_Level: float: No parameter help available
- Rl_Per: float: The reverse link packet error rate (for the selected carrier) . Range: 0 % to 100 %
- Tt_Macp_Kts_Errors: float: No parameter help available
- Tt_Conf_Level: float: No parameter help available
- Tt_Per: float: The rate of MAC packets the R&S CMW failed to receive successfully (on the selected carrier) . This result is available for physical layer subtypes 2 and 3 only. Range: 0 % to 100 %
- Mac_Packets_Sent: float: No parameter help available
- All_Macp_Kts_Trans: float: No parameter help available
- All_Macp_Kts_Error: float: No parameter help available
- All_Rl_Conf_Level: float: No parameter help available
- All_Rl_Per: float: No parameter help available

- All_Ttp_Kts_Errors: float: No parameter help available
- All_Tt_Conf_Level: float: No parameter help available
- All_Tt_Per: float: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Macp_Kts_Transm: int: No parameter help available
- Rl_Macp_Kts_Errors: int: No parameter help available
- Rl_Conf_Level: float: No parameter help available
- Rl_Per: float: The reverse link packet error rate (for the selected carrier) . Range: 0 % to 100 %
- Tt_Macp_Kts_Errors: int: No parameter help available
- Tt_Conf_Level: float: No parameter help available
- Tt_Per: float: The rate of MAC packets the R&S CMW failed to receive successfully (on the selected carrier) . This result is available for physical layer subtypes 2 and 3 only. Range: 0 % to 100 %
- Mac_Packets_Sent: int: No parameter help available
- All_Macp_Kts_Trans: int: No parameter help available
- All_Macp_Kts_Error: int: No parameter help available
- All_Rl_Conf_Level: float: No parameter help available
- All_Rl_Per: float: No parameter help available
- All_Ttp_Kts_Errors: int: No parameter help available
- All_Tt_Conf_Level: float: No parameter help available
- All_Tt_Per: float: No parameter help available

calculate(*data_rate: Optional[RsCmwEvdoSig.enums.RevLinkPerDataRate] = None*) → CalculateStruct

```
# SCPI: CALCulate:EVDO:SIGNaling<instance>:RXQuality:RLPer
value: CalculateStruct = driver.rxQuality.rlPer.calculate(data_rate = enums.
↪RevLinkPerDataRate.R0K0)


    INTRO_CMD_HELP: Returns the 'Reverse Link PER' measurement results for:

    - The specified data rate
    - The selected carrier (in multi-carrier scenarios, see 'Reverse Link␣
↪Packet Error Rate Measurement')
```

Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

param data_rate R0K0 | R19K2 | R38K4 | R76K8 | R115k2 | R153k6 | R230k4 | R307k2 | R460k8 | R614k4 | R921k6 | R1228k8 | R1843k2 | TOTal This query parameter specifies the data rate for which the results are returned, i.e. it selects a row

in the 'Reverse Link PER' result table. If it is omitted, the command returns the TOTal results, i.e. the aggregates over all data rates. Note that the 'composite' return values are aggregates over all carriers and data rates, i.e. no matter which data rate is specified, always the TOTal values are returned.

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

fetch(*data_rate: Optional[RsCmwEvdoSig.enums.RevLinkPerDataRate] = None*) → ResultData

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:RLPer
value: ResultData = driver.rxQuality.rlPer.fetch(data_rate = enums.
↪RevLinkPerDataRate.R0K0)


    INTRO_CMD_HELP: Returns the 'Reverse Link PER' measurement results for:

    - The specified data rate
    - The selected carrier (in multi-carrier scenarios, see 'Reverse Link␣
↪Packet Error Rate Measurement')
```

Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

param data_rate R0K0 | R19K2 | R38K4 | R76K8 | R115k2 | R153k6 | R230k4 | R307k2 | R460k8 | R614k4 | R921k6 | R1228k8 | R1843k2 | TOTal This query parameter specifies the data rate for which the results are returned, i.e. it selects a row in the 'Reverse Link PER' result table. If it is omitted, the command returns the TOTal results, i.e. the aggregates over all data rates. Note that the 'composite' return values are aggregates over all carriers and data rates, i.e. no matter which data rate is specified, always the TOTal values are returned.

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

read(*data_rate: Optional[RsCmwEvdoSig.enums.RevLinkPerDataRate] = None*) → ResultData

```
# SCPI: READ:EVDO:SIGNaling<instance>:RXQuality:RLPer
value: ResultData = driver.rxQuality.rlPer.read(data_rate = enums.
↪RevLinkPerDataRate.R0K0)


    INTRO_CMD_HELP: Returns the 'Reverse Link PER' measurement results for:

    - The specified data rate
    - The selected carrier (in multi-carrier scenarios, see 'Reverse Link␣
↪Packet Error Rate Measurement')
```

Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

param data_rate R0K0 | R19K2 | R38K4 | R76K8 | R115k2 | R153k6 | R230k4 | R307k2 | R460k8 | R614k4 | R921k6 | R1228k8 | R1843k2 | TOTal This query pa-

rameter specifies the data rate for which the results are returned, i.e. it selects a row in the 'Reverse Link PER' result table. If it is omitted, the command returns the TOTal results, i.e. the aggregates over all data rates. Note that the 'composite' return values are aggregates over all carriers and data rates, i.e. no matter which data rate is specified, always the TOTal values are returned.

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.rxQuality.rlPer.clone()
```

Subgroups

7.10.2.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:RLPer:STATe
```

class State

State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → str

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:RLPer:STATe
value: str = driver.rxQuality.rlPer.state.fetch()
```

Returns a string containing status information about the measurement.

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

return status: See table below

7.10.2.2 Cstate

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:RLPer:CSTate
```

class Cstate

Cstate commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → List[RsCmwEvdoSig.enums.CarrierStatus]

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:RLPer:CSTate
value: List[enums.CarrierStatus] = driver.rxQuality.rlPer.cstate.fetch()
```

Returns status information about the carrier.

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

return carrier_status: OK | VIOLated | STALe | INACtive

7.10.3 FlPerformance

SCPI Commands

```
READ:EVDO:SIGNaling<Instance>:RXQuality:FLPFormance
FETCh:EVDO:SIGNaling<Instance>:RXQuality:FLPFormance
CALCulate:EVDO:SIGNaling<Instance>:RXQuality:FLPFormance
```

class FlPerformance

FlPerformance commands group definition. 5 total commands, 2 Sub-groups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Mac_Pack_Received: float: MAC packets received Range: 0 or higher
- Phy_Packet_Slots: float: Physical packet slots Range: 0 or higher
- Th_Vs_Test_Time: float: Throughput vs. test time Range: 0 kbit/s or higher
- Th_Vs_Transm_Slots: float: Throughput vs. transmitted slots Range: 0 kbit/s or higher
- Test_Time: float: Test time Range: 0 to 10000, Unit: no. of CDMA frames
- All_Th_Vs_Test_Time: float: No parameter help available
- All_Th_Vs_Tra_Slots: float: No parameter help available

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Mac_Pack_Received: int: MAC packets received Range: 0 or higher
- Phy_Packet_Slots: int: Physical packet slots Range: 0 or higher
- Th_Vs_Test_Time: float: Throughput vs. test time Range: 0 kbit/s or higher
- Th_Vs_Transm_Slots: float: Throughput vs. transmitted slots Range: 0 kbit/s or higher
- Test_Time: int: Test time Range: 0 to 10000, Unit: no. of CDMA frames
- All_Th_Vs_Test_Time: float: No parameter help available
- All_Th_Vs_Tra_Slots: float: No parameter help available

calculate(*packet_size: Optional[RsCmwEvdoSig.enums.PacketSize] = None*) → CalculateStruct

```
# SCPI: CALCulate:EVDO:SIGNaling<instance>:RXQuality:FLPFormance
value: CalculateStruct = driver.rxQuality.flPerformance.calculate(packet_size =␣
↪enums.PacketSize.S128)
```

Returns the results of the 'Forward Link Throughput' measurement for the specified packet size (and the selected carrier in multi-carrier scenarios) , see 'Forward Link Throughput Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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.

param packet_size S128 | S256 | S512 | S1K | S2K | S3K | S4K | S5K | S6K | S7K | S8K | TOTal In bit: 128, 256, 512, 1024, 2048, 3072, 4096, 5120, 6144, 7162, 8192 This query parameter represents the physical layer packet size for which the results are returned, i.e. it selects a row in the 'Forward Link Throughput' result table. If this parameter is omitted, the command returns the TOTal results, i.e. the aggregates over all packet sizes. Note that the 'composite' return values are aggregates over all carriers and packet sizes, i.e. no matter which packet size is specified, always the TOTal values are returned.

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

fetch(*packet_size: Optional[RsCmwEvdoSig.enums.PacketSize] = None*) → ResultData

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:FLPFormance
value: ResultData = driver.rxQuality.flPerformance.fetch(packet_size = enums.
↪PacketSize.S128)
```

Returns the results of the 'Forward Link Throughput' measurement for the specified packet size (and the selected carrier in multi-carrier scenarios) , see 'Forward Link Throughput Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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.

param packet_size S128 | S256 | S512 | S1K | S2K | S3K | S4K | S5K | S6K | S7K | S8K | TOTal In bit: 128, 256, 512, 1024, 2048, 3072, 4096, 5120, 6144, 7162, 8192 This query parameter represents the physical layer packet size for which the results are returned, i.e. it selects a row in the 'Forward Link Throughput' result table. If this parameter is omitted, the command returns the TOTal results, i.e. the aggregates over all packet sizes. Note that the 'composite' return values are aggregates over all carriers and packet sizes, i.e. no matter which packet size is specified, always the TOTal values are returned.

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

read(*packet_size: Optional[RsCmwEvdoSig.enums.PacketSize] = None*) → ResultData

```
# SCPI: READ:EVDO:SIGNaling<instance>:RXQuality:FLPFormance
value: ResultData = driver.rxQuality.flPerformance.read(packet_size = enums.
↪PacketSize.S128)
```

Returns the results of the 'Forward Link Throughput' measurement for the specified packet size (and the selected carrier in multi-carrier scenarios) , see 'Forward Link Throughput Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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.

param packet_size S128 | S256 | S512 | S1K | S2K | S3K | S4K | S5K | S6K | S7K | S8K | TOTal In bit: 128, 256, 512, 1024, 2048, 3072, 4096, 5120, 6144, 7162, 8192 This query parameter represents the physical layer packet size for which the results are returned, i.e. it selects a row in the 'Forward Link Throughput' result table. If this parameter is omitted, the command returns the TOTal results, i.e. the aggregates over all packet sizes. Note that the 'composite' return values are aggregates over all carriers

and packet sizes, i.e. no matter which packet size is specified, always the TOTal values are returned.

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.rxQuality.flPerformance.clone()
```

Subgroups

7.10.3.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:FLPFormance:STATe
```

class State

State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → str

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:FLPFormance:STATe
value: str = driver.rxQuality.flPerformance.state.fetch()
```

Returns a string containing status information about the measurement.

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

return status: See table below

7.10.3.2 Cstate

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:FLPFormance:CSTate
```

class Cstate

Cstate commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → List[RsCmwEvdoSig.enums.CarrierStatus]

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:FLPFormance:CSTate
value: List[enums.CarrierStatus] = driver.rxQuality.flPerformance.cstate.fetch()
```

Returns status information about the carrier.

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

return carrier_status: OK | VIOLated | STALe | INACtive

7.10.4 RlPerformance

SCPI Commands

```
READ:EVDO:SIGNaling<Instance>:RXQuality:RLPFormance
FETCh:EVDO:SIGNaling<Instance>:RXQuality:RLPFormance
CALCulate:EVDO:SIGNaling<Instance>:RXQuality:RLPFormance
```

class RlPerformance

RlPerformance commands group definition. 5 total commands, 2 Sub-groups, 3 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Packet_Size: str: The packet size given as string representations of the enum constants S128 | S256 | S512 | S1K | S2K | S3K | S4K | S5K | S6K | S7K | S8K | TOTal corresponding to the data rates in bit: 128, 256, 512, 1024, 2048, 3072, 4096, 5120, 6144, 7162, 8192.
- Mac_Pack_Received: float: The number of MAC packets successfully received by the R&S CMW (on the selected carrier) . Range: 0 to 10E+3
- Th_Vs_Test_Time: float: The average throughput in kbit/s (on the selected carrier) during the test time Range: 0 kbit/s to 99.99999E+3 kbit/s
- Test_Time: float: The elapsed test time as the number of 26.67 ms CDMA frames. Range: 0 to 10E+3
- All_Th_Vs_Test_Time: float: The average throughput in kbit/s (on the selected carrier) during the test time Range: 0 kbit/s to 99.99999E+3 kbit/s

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Packet_Size: str: The packet size given as string representations of the enum constants S128 | S256 | S512 | S1K | S2K | S3K | S4K | S5K | S6K | S7K | S8K | TOTal corresponding to the data rates in bit: 128, 256, 512, 1024, 2048, 3072, 4096, 5120, 6144, 7162, 8192.
- Mac_Pack_Received: int: The number of MAC packets successfully received by the R&S CMW (on the selected carrier) . Range: 0 to 10E+3
- Th_Vs_Test_Time: float: The average throughput in kbit/s (on the selected carrier) during the test time Range: 0 kbit/s to 99.99999E+3 kbit/s
- Test_Time: int: The elapsed test time as the number of 26.67 ms CDMA frames. Range: 0 to 10E+3
- All_Th_Vs_Test_Time: float: The average throughput in kbit/s (on the selected carrier) during the test time Range: 0 kbit/s to 99.99999E+3 kbit/s

calculate(*data_rate: Optional[RsCmwEvdoSig.enums.RevLinkPerDataRate] = None*) → CalculateStruct

```
# SCPI: CALCulate:EVDO:SIGNaling<instance>:RXQuality:RLPFormance
value: CalculateStruct = driver.rxQuality.rlPerformance.calculate(data_rate =␣
↪enums.RevLinkPerDataRate.R0K0)
```

Returns the results of the 'Reverse Link Throughput' measurement for the specified data rate (and the selected carrier in multi-carrier scenarios) , see 'Reverse Link Throughput Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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.

param data_rate R0K0 | R19K2 | R38K4 | R76K8 | R115k2 | R153k6 | R230k4 | R307k2 | R460k8 | R614k4 | R921k6 | R1228k8 | R1843k2 | TOTal This query parameter specifies the data rate for which the results are returned, i.e. it selects a row in the 'Reverse Link Throughput' result table. If omitted, the TOTal values are returned, i.e. the aggregates over all data rates. Note that the 'composite' return values are aggregates over all carriers and data rates, i.e. no matter which data rate is specified, always the TOTal values are returned.

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

fetch(*data_rate: Optional[RsCmwEvdoSig.enums.RevLinkPerDataRate] = None*) → ResultData

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:RLPFormance
value: ResultData = driver.rxQuality.rlPerformance.fetch(data_rate = enums.
↪RevLinkPerDataRate.R0K0)
```

Returns the results of the 'Reverse Link Throughput' measurement for the specified data rate (and the selected carrier in multi-carrier scenarios) , see 'Reverse Link Throughput Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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.

param data_rate R0K0 | R19K2 | R38K4 | R76K8 | R115k2 | R153k6 | R230k4 | R307k2 | R460k8 | R614k4 | R921k6 | R1228k8 | R1843k2 | TOTal This query parameter specifies the data rate for which the results are returned, i.e. it selects a row in the 'Reverse Link Throughput' result table. If omitted, the TOTal values are returned, i.e. the aggregates over all data rates. Note that the 'composite' return values are aggregates over all carriers and data rates, i.e. no matter which data rate is specified, always the TOTal values are returned.

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

read(*data_rate: Optional[RsCmwEvdoSig.enums.RevLinkPerDataRate] = None*) → ResultData

```
# SCPI: READ:EVDO:SIGNaling<instance>:RXQuality:RLPFormance
value: ResultData = driver.rxQuality.rlPerformance.read(data_rate = enums.
↪RevLinkPerDataRate.R0K0)
```

Returns the results of the 'Reverse Link Throughput' measurement for the specified data rate (and the selected carrier in multi-carrier scenarios) , see 'Reverse Link Throughput Measurement'. Preselect the related carrier using the method RsCmwEvdoSig.Configure.Carrier.setting command. 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.

param data_rate R0K0 | R19K2 | R38K4 | R76K8 | R115k2 | R153k6 | R230k4 | R307k2 | R460k8 | R614k4 | R921k6 | R1228k8 | R1843k2 | TOTal This query parameter specifies the data rate for which the results are returned, i.e. it selects a row in the 'Reverse Link Throughput' result table. If omitted, the TOTal values are returned, i.e. the aggregates over all data rates. Note that the 'composite' return values are aggregates over all carriers and data rates, i.e. no matter which data rate is specified, always the TOTal values are returned.

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.rxQuality.rlPerformance.clone()
```

Subgroups

7.10.4.1 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:RLPFormance:STATe
```

class State

State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → str

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:RLPFormance:STATe
value: str = driver.rxQuality.rlPerformance.state.fetch()
```

Returns a string containing status information about the measurement.

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

return status: See table below

7.10.4.2 Cstate

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:RLPFormance:CSTate
```

class Cstate

Cstate commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → List[RsCmwEvdoSig.enums.CarrierStatus]

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:RLPFormance:CSTate
value: List[enums.CarrierStatus] = driver.rxQuality.rlPerformance.cstate.fetch()
```

Returns status information about the carrier.

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

return carrier_status: OK | VIOLated | STALe | INACtive

7.10.5 State

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwEvdoSig.enums.ResourceState

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:STATe
value: enums.ResourceState = driver.rxQuality.state.fetch()
```

No command help available

return state: No help available

Cloning the Group

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

Subgroups

7.10.5.1 All

SCPI Commands

```
FETCh:EVDO:SIGNaling<Instance>:RXQuality:STATe:ALL
```

class All

All commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: No parameter help available
- Sync_State: enums.ResourceState: No parameter help available
- Resource_State: enums.ResourceState: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:EVDO:SIGNaling<instance>:RXQuality:STATe:ALL
value: FetchStruct = driver.rxQuality.state.all.fetch()
```

No command help available

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

INDEX

F

I

R

S