RsCmwWcdmaSig

Release 3.8.10.26

Rohde & Schwarz

May 27, 2021

CONTENTS:


CMW

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

RsCmwWcdmaSig 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

RsCmwWcdmaSig 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 RsCmwWcdmaSig`

Option 2 - Installing in Pycharm

- In Pycharm Menu `File->Settings->Project->Project Interpreter` click on the **'+'** button on the bottom left
- Type `RsCmwWcdmaSig` 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 RsCmwWcdmaSig offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwWcdmaSig needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwWcdmaSig package to your computer from the pypi.org: https://pypi.org/project/RsCmwWcdmaSig/#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\RsCmwWcdmaSig-3.8.10.26.tar`

1.3 Finding Available Instruments

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

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

from RsCmwWcdmaSig import *

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

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

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

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

from RsCmwWcdmaSig import *

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

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

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

from RsCmwWcdmaSig import *

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

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

from RsCmwWcdmaSig import *

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

More `option_string` tokens are separated by comma:

```
driver = RsCmwWcdmaSig('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 RsCmwWcdmaSig objects:

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

from RsCmwWcdmaSig import *

driver1 = RsCmwWcdmaSig('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsCmwWcdmaSig.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 RsCmwWcdmaSig 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 RsCmwWcdmaSig import *

driver = RsCmwWcdmaSig('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 RsCmwWcdmaSig import *

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

driver = RsCmwWcdmaSig('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

RsCmwWcdmaSig 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 RsCmwWcdmaSig 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 RsCmwWcdmaSig import *

```

(continues on next page)

(continued from previous page)

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

import threading
from RsCmwWcdmaSig import *
```

(continues on next page)

(continued from previous page)

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


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

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsCmwWcdmaSig import *


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

threads = []
```

(continues on next page)

(continued from previous page)

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

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

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

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

Multiple instrument sessions accessed from multiple threads

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

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

Run the following example:

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

import threading
from RsCmwWcdmaSig import *


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

(continues on next page)

(continued from previous page)

```
driver2.utilities.visa_timeout = 200

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

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

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

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

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

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 AckNack

```
# Example value:
value = enums.AckNack.ACK
# All values (3x):
ACK | DTX | NACK
```

3.2 AmrCodecModeNarrow

```
# First value:
value = enums.AmrCodecModeNarrow.A
# Last value:
value = enums.AmrCodecModeNarrow.M
# All values (9x):
A | B | C | D | E | F | G | H
M
```

3.3 AmrCodecModeWide

```
# First value:
value = enums.AmrCodecModeWide.A
# Last value:
value = enums.AmrCodecModeWide.M2
# All values (12x):
A | B | C | D | E | F | G | H
I | M | M1 | M2
```

3.4 AutoManualMode

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

3.5 AveragingMode

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

3.6 BandClass

```
# First value:
value = enums.BandClass.AWS
# Last value:
value = enums.BandClass.USPC
# All values (21x):
AWS | B18M | IEXT | IM2K | JTAC | KCEL | KPCS | LO7C
N45T | NA7C | NA8S | NA9C | NAPC | PA4M | PA8M | PS7C
TACS | U25B | U25F | USC | USPC
```

3.7 BasebandBoard

```
# First value:
value = enums.BasebandBoard.BBR1
# Last value:
value = enums.BasebandBoard.SUW44
# All values (140x):
BBR1 | BBR11 | BBR12 | BBR13 | BBR14 | BBR2 | BBR21 | BBR22
BBR23 | BBR24 | BBR3 | BBR31 | BBR32 | BBR33 | BBR34 | BBR4
BBR41 | BBR42 | BBR43 | BBR44 | BBT1 | BBT11 | BBT12 | BBT13
BBT14 | BBT2 | BBT21 | BBT22 | BBT23 | BBT24 | BBT3 | BBT31
BBT32 | BBT33 | BBT34 | BBT4 | BBT41 | BBT42 | BBT43 | BBT44
SUA012 | SUA034 | SUA056 | SUA078 | SUA1 | SUA11 | SUA112 | SUA12
SUA13 | SUA134 | SUA14 | SUA15 | SUA156 | SUA16 | SUA17 | SUA178
SUA18 | SUA2 | SUA21 | SUA212 | SUA22 | SUA23 | SUA234 | SUA24
SUA25 | SUA256 | SUA26 | SUA27 | SUA278 | SUA28 | SUA3 | SUA31
SUA312 | SUA32 | SUA33 | SUA334 | SUA34 | SUA35 | SUA356 | SUA36
SUA37 | SUA378 | SUA38 | SUA4 | SUA41 | SUA412 | SUA42 | SUA43
SUA434 | SUA44 | SUA45 | SUA456 | SUA46 | SUA47 | SUA478 | SUA48
SUA5 | SUA6 | SUA7 | SUA8 | SUU1 | SUU11 | SUU12 | SUU13
```

(continues on next page)

(continued from previous page)

```
SUU14 | SUU2 | SUU21 | SUU22 | SUU23 | SUU24 | SUU3 | SUU31
SUU32 | SUU33 | SUU34 | SUU4 | SUU41 | SUU42 | SUU43 | SUU44
SUW1 | SUW11 | SUW12 | SUW13 | SUW14 | SUW2 | SUW21 | SUW22
SUW23 | SUW24 | SUW3 | SUW31 | SUW32 | SUW33 | SUW34 | SUW4
SUW41 | SUW42 | SUW43 | SUW44
```

3.8 BitPattern

```
# Example value:
value = enums.BitPattern.ALL0
# All values (7x):
ALL0 | ALL1 | ALTernating | PRBS11 | PRBS13 | PRBS15 | PRBS9
```

3.9 Bsic

```
# Example value:
value = enums.Bsic.NONVerified
# All values (2x):
NONVerified | VERified
```

3.10 BtfdDataRate

```
# First value:
value = enums.BtfdDataRate.R10K2
# Last value:
value = enums.BtfdDataRate.R7K95
# All values (9x):
R10K2 | R12K2 | R1K95 | R4K75 | R5K15 | R5K9 | R6K7 | R7K4
R7K95
```

3.11 CallRelease

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

3.12 CbsMessageSeverity

```
# First value:
value = enums.CbsMessageSeverity.AAMBer
# Last value:
value = enums.CbsMessageSeverity.UDEFined
# All values (9x):
AAMBer | AEXTreme | APResidentia | ASEVere | EARThquake | ETWarning | ETWTest | TSUNami
UDEFined
```

3.13 CellConfig

```
# First value:
value = enums.CellConfig._3CHS
# Last value:
value = enums.CellConfig.WCDMa
# All values (16x):
_3CHS | _3DUPlus | _3HDU | _4CHS | _4DUPlus | _4HDU | DCHS | DDUPlus
DHDU | HDUPlus | HSDPa | HSPA | HSPLus | HSUPa | QPSK | WCDMa
```

3.14 CellPower

```
# Example value:
value = enums.CellPower.NAV
# All values (4x):
NAV | OFL | OK | UFL
```

3.15 ChannelType

```
# Example value:
value = enums.ChannelType.CQI
# All values (3x):
CQI | FIXed | UDEFined
```

3.16 Cipher

```
# Example value:
value = enums.Cipher.UEA0
# All values (3x):
UEA0 | UEA1 | UEA2
```

3.17 ClosedLoopPower

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

3.18 CmodeActivation

```
# Example value:
value = enums.CmodeActivation.MEASurement
# All values (2x):
MEASurement | RAB
```

3.19 CmodePatternSelection

```
# Example value:
value = enums.CmodePatternSelection.NONE
# All values (4x):
NONE | SINGle | UEReport | ULCM
```

3.20 CmSerRejectType

```
# Example value:
value = enums.CmSerRejectType.ECALl
# All values (7x):
ECALl | ECSMs | NCALl | NCECall | NCSMs | NESMs | SMS
```

3.21 CodingGroup

```
# Example value:
value = enums.CodingGroup.DCMClass
# All values (3x):
DCMClass | GDCoding | REServed
```

3.22 CompressedMode

```
# Example value:
value = enums.CompressedMode.NN
# All values (4x):
NN | NY | YN | YY
```

3.23 CompressedModeBand

```
# First value:
value = enums.CompressedModeBand.OB1
# Last value:
value = enums.CompressedModeBand.OB9
# All values (25x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16
OB17 | OB18 | OB19 | OB2 | OB20 | OB21 | OB22 | OB25
OB26 | OB3 | OB32 | OB4 | OB5 | OB6 | OB7 | OB8
OB9
```

3.24 Condition

```
# Example value:
value = enums.Condition.ALTernating
# All values (5x):
ALTernating | MAXPower | MINPower | NONE | TPOWer
```

3.25 ConditionB

```
# Example value:
value = enums.ConditionB.ALTernating
# All values (4x):
ALTernating | MAXPower | MINPower | TPOWer
```

3.26 CounterValue

```
# Example value:
value = enums.CounterValue.N1
# All values (8x):
N1 | N10 | N100 | N2 | N20 | N200 | N4 | N50
```

3.27 Cqi

```
# First value:
value = enums.Cqi._0
# Last value:
value = enums.Cqi.DTX
# All values (32x):
_0 | _1 | _10 | _11 | _12 | _13 | _14 | _15
_16 | _17 | _18 | _19 | _2 | _20 | _21 | _22
_23 | _24 | _25 | _26 | _27 | _28 | _29 | _3
_30 | _4 | _5 | _6 | _7 | _8 | _9 | DTX
```

3.28 CsFallbackConnectionType

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

3.29 CswitchedAction

```
# Example value:
value = enums.CswitchedAction.CONNect
# All values (5x):
CONNect | DISConnect | HANDover | SSMS | UNRegister
```

3.30 CswitchedState

```
# First value:
value = enums.CswitchedState.ALERting
# Last value:
value = enums.CswitchedState.SIGNaling
# All values (15x):
ALERting | CESTablished | CONNecting | IHANdover | IHPReparate | IREDirection |␣
↪IRPReparate | OFF
OHANdover | ON | OREDirection | PAGing | REGister | RELeasing | SIGNaling
```

3.31 CurrentConnectionType

```
# Example value:
value = enums.CurrentConnectionType.NONE
# All values (8x):
NONE | PACKet | SRB | TEST | VIDeo | VIPacket | VOICe | VOPacket
```

3.32 DataRateDownlink

```
# Example value:
value = enums.DataRateDownlink.HSDPa
# All values (7x):
HSDPa | R128 | R16 | R32 | R384 | R64 | R8
```

3.33 DataRateUplink

```
# Example value:
value = enums.DataRateUplink.HSUPa
# All values (7x):
HSUPa | R128 | R16 | R32 | R384 | R64 | R8
```

3.34 DchEnhanced

```
# Example value:
value = enums.DchEnhanced.BASic
# All values (3x):
BASic | FULL | NO
```

3.35 DestinationState

```
# Example value:
value = enums.DestinationState.CPCH
# All values (4x):
CPCH | FACH | IDLE | UPCH
```

3.36 DsTime

```
# Example value:
value = enums.DsTime.OFF
# All values (4x):
OFF | ON | P1H | P2H
```

3.37 EhichIndicatorMode

```
# Example value:
value = enums.EhichIndicatorMode.ACK
# All values (5x):
ACK | ALTernating | CRC | DTX | NACK
```

3.38 Enable

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

3.39 ErgchIndicatorMode

```
# Example value:
value = enums.ErgchIndicatorMode.ALTernating
# All values (7x):
ALTernating | CONTinuous | DOWN | DTX | HARQ | SINGle | UP
```

3.40 Etfci

```
# First value:
value = enums.Etfci._0
# Last value:
value = enums.Etfci.DTX
# All values (129x):
_0 | _1 | _10 | _100 | _101 | _102 | _103 | _104
_105 | _106 | _107 | _108 | _109 | _11 | _110 | _111
_112 | _113 | _114 | _115 | _116 | _117 | _118 | _119
_12 | _120 | _121 | _122 | _123 | _124 | _125 | _126
_127 | _13 | _14 | _15 | _16 | _17 | _18 | _19
_2 | _20 | _21 | _22 | _23 | _24 | _25 | _26
_27 | _28 | _29 | _3 | _30 | _31 | _32 | _33
_34 | _35 | _36 | _37 | _38 | _39 | _4 | _40
```

(continues on next page)

(continued from previous page)

```
_41 | _42 | _43 | _44 | _45 | _46 | _47 | _48
_49 | _5 | _50 | _51 | _52 | _53 | _54 | _55
_56 | _57 | _58 | _59 | _6 | _60 | _61 | _62
_63 | _64 | _65 | _66 | _67 | _68 | _69 | _7
_70 | _71 | _72 | _73 | _74 | _75 | _76 | _77
_78 | _79 | _8 | _80 | _81 | _82 | _83 | _84
_85 | _86 | _87 | _88 | _89 | _9 | _90 | _91
_92 | _93 | _94 | _95 | _96 | _97 | _98 | _99
DTX
```

3.41 Fader

```
# First value:
value = enums.Fader.FAD012
# Last value:
value = enums.Fader.FAD8
# All values (60x):
FAD012 | FAD034 | FAD056 | FAD078 | FAD1 | FAD11 | FAD112 | FAD12
FAD13 | FAD134 | FAD14 | FAD15 | FAD156 | FAD16 | FAD17 | FAD178
FAD18 | FAD2 | FAD21 | FAD212 | FAD22 | FAD23 | FAD234 | FAD24
FAD25 | FAD256 | FAD26 | FAD27 | FAD278 | FAD28 | FAD3 | FAD31
FAD312 | FAD32 | FAD33 | FAD334 | FAD34 | FAD35 | FAD356 | FAD36
FAD37 | FAD378 | FAD38 | FAD4 | FAD41 | FAD412 | FAD42 | FAD43
FAD434 | FAD44 | FAD45 | FAD456 | FAD46 | FAD47 | FAD478 | FAD48
FAD5 | FAD6 | FAD7 | FAD8
```

3.42 FadingStandard

```
# First value:
value = enums.FadingStandard.B261
# Last value:
value = enums.FadingStandard.VA30
# All values (19x):
B261 | B262 | B263 | BDEath | C1 | C2 | C3 | C4
C5 | C6 | C8 | HST | MPRopagation | PA3 | PB3 | USER
VA12 | VA3 | VA30
```

3.43 FilledBlocks

```
# First value:
value = enums.FilledBlocks.P0031
# Last value:
value = enums.FilledBlocks.P1000
# All values (17x):
P0031 | P0033 | P0036 | P0038 | P0042 | P0045 | P0050 | P0056
```

(continues on next page)

(continued from previous page)

```
P0062 | P0071 | P0083 | P0100 | P0125 | P0167 | P0250 | P0500
P1000
```

3.44 GapSize

```
# Example value:
value = enums.GapSize.ANY
# All values (3x):
ANY | M10 | M5
```

3.45 GeneratorState

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

3.46 GeoScope

```
# Example value:
value = enums.GeoScope.CIMMediate
# All values (4x):
CIMMediate | CNORmal | PLMN | SERVice
```

3.47 GsmBand

```
# Example value:
value = enums.GsmBand.G04
# All values (5x):
G04 | G085 | G09 | G18 | G19
```

3.48 Handover

```
# Example value:
value = enums.Handover.PACKet
# All values (3x):
PACKet | TM | VOICe
```

3.49 HappyBit

```
# Example value:
value = enums.HappyBit.DTX
# All values (3x):
DTX | HAPPy | UNHappy
```

3.50 HoverExtDestination

```
# Example value:
value = enums.HoverExtDestination.CDMA
# All values (5x):
CDMA | EVDO | GSM | LTE | WCDMa
```

3.51 HrVersion

```
# Example value:
value = enums.HrVersion.RV0
# All values (2x):
RV0 | TABLe
```

3.52 HsdpaModulation

```
# Example value:
value = enums.HsdpaModulation.Q16
# All values (3x):
Q16 | Q64 | QPSK
```

3.53 HsetFixed

```
# First value:
value = enums.HsetFixed.H1AI
# Last value:
value = enums.HsetFixed.HCMT
# All values (45x):
H1AI | H1BI | H1CI | H1M1 | H1M2 | H1MI | H2M1 | H2M2
H3A1 | H3A2 | H3B1 | H3B2 | H3C1 | H3C2 | H3M1 | H3M2
H4M1 | H5M1 | H6A1 | H6A2 | H6B1 | H6B2 | H6C1 | H6C2
H6M1 | H6M2 | H8A3 | H8AI | H8B3 | H8BI | H8C3 | H8CI
H8M3 | H8MI | H8MT | HAA1 | HAA2 | HAB1 | HAB2 | HAC1
HAC2 | HAM1 | HAM2 | HCM1 | HCMT
```

3.54 HspaTestModeDirection

```
# Example value:
value = enums.HspaTestModeDirection.HSDPa
# All values (2x):
HSDPa | HSPA
```

3.55 HsScchType

```
# Example value:
value = enums.HsScchType.AUTomatic
# All values (6x):
AUTomatic | CH1 | CH2 | CH3 | CH4 | RANDom
```

3.56 HsupaModulation

```
# Example value:
value = enums.HsupaModulation.Q16
# All values (2x):
Q16 | QPSK
```

3.57 InsertLossMode

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

3.58 IpAddrIndex

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

3.59 LevelSeqState

```
# Example value:
value = enums.LevelSeqState.FAILed
# All values (5x):
FAILed | IDLE | RUNNing | SCHanged | SCONflict
```

3.60 LogCategory

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

3.61 LongSmsHandling

```
# Example value:
value = enums.LongSmsHandling.MSMS
# All values (2x):
MSMS | TRUNcate
```

3.62 LteBand

```
# First value:
value = enums.LteBand.OB1
# Last value:
value = enums.LteBand.UDEFined
# All values (68x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16
OB17 | OB18 | OB19 | OB2 | OB20 | OB21 | OB22 | OB23
OB24 | OB25 | OB250 | OB252 | OB255 | OB26 | OB27 | OB28
OB29 | OB3 | OB30 | OB31 | OB32 | OB33 | OB34 | OB35
OB36 | OB37 | OB38 | OB39 | OB4 | OB40 | OB41 | OB42
OB43 | OB44 | OB45 | OB46 | OB48 | OB49 | OB5 | OB50
OB51 | OB52 | OB6 | OB65 | OB66 | OB67 | OB68 | OB69
OB7 | OB70 | OB71 | OB72 | OB73 | OB74 | OB75 | OB76
OB8 | OB85 | OB9 | UDEFined
```

3.63 MaxChanCode

```
# Example value:
value = enums.MaxChanCode.S16
# All values (8x):
S16 | S22 | S224 | S24 | S32 | S4 | S64 | S8
```

3.64 MaxRelVersion

```
# First value:
value = enums.MaxRelVersion.AUTO
# Last value:
value = enums.MaxRelVersion.R99
# All values (9x):
AUTO | R10 | R11 | R5 | R6 | R7 | R8 | R9
R99
```

3.65 MeasType

```
# Example value:
value = enums.MeasType.GENeral
# All values (2x):
GENeral | MISSed
```

3.66 MessageClass

```
# Example value:
value = enums.MessageClass.CL0
# All values (5x):
CL0 | CL1 | CL2 | CL3 | NONE
```

3.67 MessageHandling

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

3.68 MobilityMode

```
# Example value:
value = enums.MobilityMode.CCORder
# All values (4x):
CCORder | HANDover | NAV | REDirection
```

3.69 MonitoredHarq

```
# First value:
value = enums.MonitoredHarq.ALL
# Last value:
value = enums.MonitoredHarq.H7
# All values (9x):
ALL | H0 | H1 | H2 | H3 | H4 | H5 | H6
H7
```

3.70 NetworkAndGps

```
# Example value:
value = enums.NetworkAndGps.BOTH
# All values (4x):
BOTH | NETWork | NONE | UE
```

3.71 NominalPowerMode

```
# Example value:
value = enums.NominalPowerMode.AUToranging
# All values (3x):
AUToranging | MANual | ULPC
```

3.72 NrOfDigits

```
# Example value:
value = enums.NrOfDigits.D2
# All values (2x):
D2 | D3
```

3.73 NtOperMode

```
# Example value:
value = enums.NtOperMode.M1
# All values (2x):
M1 | M2
```

3.74 OcnsChannelType

```
# Example value:
value = enums.OcnsChannelType.AUTO
# All values (5x):
AUTO | R5 | R6 | R7 | R99
```

3.75 OperationBand

```
# First value:
value = enums.OperationBand.OB1
# Last value:
value = enums.OperationBand.UDEFined
# All values (30x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16
OB17 | OB18 | OB19 | OB2 | OB20 | OB21 | OB22 | OB25
OB26 | OB3 | OB32 | OB4 | OB5 | OB6 | OB7 | OB8
OB9 | OBL1 | OBS1 | OBS2 | OBS3 | UDEFined
```

3.76 OperBandConfig

```
# Example value:
value = enums.OperBandConfig.C1
# All values (7x):
C1 | C2 | C3 | C4 | C5 | C6 | UDEFined
```

3.77 ParameterType

```
# Example value:
value = enums.ParameterType.PRIMary
# All values (2x):
PRIMary | SECondary
```

3.78 PatternType

```
# Example value:
value = enums.PatternType.DU
# All values (2x):
DU | UD
```

3.79 PhaseReference

```
# Example value:
value = enums.PhaseReference.PCPich
# All values (2x):
PCPich | SCPich
```

3.80 PowerControlMode

```
# Example value:
value = enums.PowerControlMode.M0
# All values (4x):
M0 | M1 | OFF | ON
```

3.81 PowerStrategy

```
# Example value:
value = enums.PowerStrategy.AF
# All values (3x):
AF | BF | CE
```

3.82 Priority

```
# Example value:
value = enums.Priority.BACKground
# All values (3x):
BACKground | HIGH | NORMal
```

3.83 Procedure

```
# Example value:
value = enums.Procedure.CSOPs
# All values (3x):
CSOPs | CSPS | PS
```

3.84 PswitchedAction

```
# Example value:
value = enums.PswitchedAction.ACONnect
# All values (4x):
ACONnect | CONNect | DISConnect | HANDover
```

3.85 PswitchedState

```
# First value:
value = enums.PswitchedState.ATTached
# Last value:
value = enums.PswitchedState.SIGNaling
# All values (14x):
ATTached | CESTablished | CONNecting | IHANdover | IHPReparate | IREDirection |␣
↪IRPReparate | OFF
OHANdover | ON | OREDirection | PAGing | RELeasing | SIGNaling
```

3.86 ReducedSignState

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

3.87 RefChannelDataRate

```
# Example value:
value = enums.RefChannelDataRate.BTFD
# All values (6x):
BTFD | R12K2 | R144k | R384k | R64K | R768k
```

3.88 RejectCause

```
# Example value:
value = enums.RejectCause.CSCongestion
# All values (6x):
CSCongestion | CSUNspecific | OFF | ON | PSCongestion | PSUNspecific
```

3.89 RejectionCauseA

```
# First value:
value = enums.RejectionCauseA.C100
# Last value:
value = enums.RejectionCauseA.ON
# All values (30x):
C100 | C101 | C11 | C111 | C12 | C13 | C15 | C17
C2 | C20 | C21 | C22 | C23 | C25 | C3 | C32
C33 | C34 | C38 | C4 | C48 | C5 | C6 | C95
C96 | C97 | C98 | C99 | OFF | ON
```

3.90 RejectionCauseB

```
# First value:
value = enums.RejectionCauseB.C10
# Last value:
value = enums.RejectionCauseB.ON
# All values (38x):
C10 | C100 | C101 | C11 | C111 | C12 | C13 | C14
C15 | C16 | C17 | C2 | C20 | C21 | C22 | C23
C25 | C28 | C3 | C32 | C33 | C34 | C38 | C4
C40 | C48 | C5 | C6 | C7 | C8 | C9 | C95
C96 | C97 | C98 | C99 | OFF | ON
```

3.91 Repeat

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

3.92 RepetitionB

```
# Example value:
value = enums.RepetitionB.CONTinuous
# All values (3x):
CONTinuous | ONCE | SGINit
```

3.93 ResourceState

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

3.94 ResultState

```
# Example value:
value = enums.ResultState.FAIL
# All values (3x):
FAIL | PASS | RUN
```

3.95 ResultStatus2

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

3.96 RetransmisionSeqNr

```
# Example value:
value = enums.RetransmisionSeqNr._0
# All values (5x):
_0 | _1 | _2 | _3 | DTX
```

3.97 RlcMode

```
# Example value:
value = enums.RlcMode.ACKNowledge
# All values (2x):
ACKNowledge | TRANsparent
```

3.98 RmcDomain

```
# Example value:
value = enums.RmcDomain.CS
# All values (2x):
CS | PS
```

3.99 RrcState

```
# Example value:
value = enums.RrcState.CPCH
# All values (5x):
CPCH | DCH | FACH | IDLE | UPCH
```

3.100 RvcSequence

```
# Example value:
value = enums.RvcSequence.S1
# All values (8x):
S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined
```

3.101 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
```

(continues on next page)

(continued from previous page)

```
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
```

3.102 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
```

3.103 SampleRate

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

3.104 Scenario

```
# First value:
value = enums.Scenario.DBFading
# Last value:
value = enums.Scenario.UNDefined
# All values (12x):
DBFading | DBFDiversity | DCARrier | DCFading | DCFDiversity | DCHSpa | FCHSpa | SCELl
SCFading | SCFDiversity | TCHSpa | UNDefined
```

3.105 SimCardType

```
# Example value:
value = enums.SimCardType.C2G
# All values (3x):
C2G | C3G | MILenage
```

3.106 SlopeType

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

3.107 SmsDataCoding

```
# Example value:
value = enums.SmsDataCoding.BIT7
# All values (3x):
BIT7 | BIT8 | REServed
```

3.108 SourceInt

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

3.109 SourceTime

```
# Example value:
value = enums.SourceTime.CMWTime
# All values (2x):
CMWTime | DATE
```

3.110 SrbDataRate

```
# Example value:
value = enums.SrbDataRate.R13K6
# All values (4x):
R13K6 | R1K7 | R2K5 | R3K4
```

3.111 SrbSingleType

```
# Example value:
value = enums.SrbSingleType.CDCH
# All values (2x):
CDCH | CFACh
```

3.112 StopCondition

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

3.113 SubTest

```
# Example value:
value = enums.SubTest.S1
# All values (5x):
S1 | S2 | S3 | S4 | S5
```

3.114 SucessState

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

3.115 Sync

```
# Example value:
value = enums.Sync.NAV
# All values (3x):
NAV | NOSYnc | OK
```

3.116 SyncState

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

3.117 TableIndex

```
# Example value:
value = enums.TableIndex.CONFormance
# All values (4x):
CONFormance | FIXed | FOLLow | SEQuence
```

3.118 TerminatingType

```
# Example value:
value = enums.TerminatingType.RMC
# All values (5x):
RMC | SRB | TEST | VIDeo | VOICe
```

3.119 TestCase

```
# Example value:
value = enums.TestCase.AWGN
# All values (2x):
AWGN | FADing
```

3.120 TestMode

```
# Example value:
value = enums.TestMode.HOLD
# All values (2x):
HOLD | UPDown
```

3.121 TestModeType

```
# Example value:
value = enums.TestModeType.BTFD
# All values (5x):
BTFD | FACH | HSPA | RHSPa | RMC
```

3.122 TpcMode

```
# Example value:
value = enums.TpcMode.A1S1
# All values (3x):
A1S1 | A1S2 | A2S1
```

3.123 TpcSetType

```
# First value:
value = enums.TpcSetType.ALL0
# Last value:
value = enums.TpcSetType.ULCM
# All values (19x):
ALL0 | ALL1 | ALTernating | CLOop | CONTinuous | CTFC | DHIB | MPEDch
PHDown | PHUP | SAL0 | SAL1 | SALT | TSABc | TSE | TSEF
TSF | TSGH | ULCM
```

3.124 TpcState

```
# First value:
value = enums.TpcState.ALTernating
# Last value:
value = enums.TpcState.TRANsition
# All values (14x):
ALTernating | CONTinous | FAILed | IDLE | MAXPower | MINPower | MRESource | SCHanged
SCONflict | SEARching | SINGle | TPLocked | TPUNlocked | TRANsition
```

3.125 TransGapType

```
# Example value:
value = enums.TransGapType.AF
# All values (3x):
AF | AR | B
```

3.126 TransGapTypeExtended

```
# Example value:
value = enums.TransGapTypeExtended.A
# All values (8x):
A | B | C | D | E | F | RFA | RFB
```

3.127 TransTimeInterval

```
# Example value:
value = enums.TransTimeInterval.M10
# All values (2x):
M10 | M2
```

3.128 TriggerMode

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

3.129 TtiExtended

```
# Example value:
value = enums.TtiExtended.M10
# All values (4x):
M10 | M20 | M40 | M80
```

3.130 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
```

3.131 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
```

3.132 UeAlgorithm

```
# Example value:
value = enums.UeAlgorithm.EXTRapolation
# All values (2x):
EXTRapolation | INTerpolation
```

3.133 UeNaviSupport

```
# Example value:
value = enums.UeNaviSupport.NETWork
# All values (4x):
NETWork | NONE | NUE | UE
```

3.134 UePowerClass

```
# Example value:
value = enums.UePowerClass.PC1
# All values (5x):
PC1 | PC2 | PC3 | PC3B | PC4
```

3.135 UnscheduledTransType

```
# Example value:
value = enums.UnscheduledTransType.DTX
# All values (2x):
DTX | DUMMy
```

3.136 UsedSendMethod

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

3.137 UtraMode

```
# Example value:
value = enums.UtraMode.BOTH
# All values (3x):
BOTH | FDD | TDD
```

3.138 UtranTestMode

```
# Example value:
value = enums.UtranTestMode.MODE1
# All values (3x):
MODE1 | MODE2 | OFF
```

3.139 VideoRate

```
# Example value:
value = enums.VideoRate.R64K
# All values (1x):
R64K
```

3.140 VoiceCodec

```
# Example value:
value = enums.VoiceCodec.NB
# All values (2x):
NB | WB
```

3.141 VoiceSource

```
# Example value:
value = enums.VoiceSource.LOOPback
# All values (2x):
LOOPback | SPEech
```

3.142 WizzardSelection

```
# Example value:
value = enums.WizzardSelection.DHIP
# All values (8x):
DHIP | ERGM | HCQI | HDMT | HSMT | HUMP | HUMT | OOS
```

3.143 YesNoStatus

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

3.144 Zone

```
# Example value:
value = enums.Zone.NONE
# All values (2x):
NONE | Z1
```

CHAPTER

FOUR

REPCAPS

4.1 Carrier (Global)

```
# Setting:
driver.repcap_carrier_set(repcap.Carrier.C1)
# Range:
C1 .. C32
# All values (32x):
C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8
C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16
C17 | C18 | C19 | C20 | C21 | C22 | C23 | C24
C25 | C26 | C27 | C28 | C29 | C30 | C31 | C32
```

4.2 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
```

4.3 Band

```
# First value:
value = repcap.Band.B1
# Range:
B1 .. B32
# All values (32x):
B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8
B9 | B10 | B11 | B12 | B13 | B14 | B15 | B16
B17 | B18 | B19 | B20 | B21 | B22 | B23 | B24
B25 | B26 | B27 | B28 | B29 | B30 | B31 | B32
```

4.4 BandCombination

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

4.5 Cell

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

4.6 CounterNo

```
# First value:
value = repcap.CounterNo.Nr313
# Values (1x):
Nr313
```

4.7 Cycle

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

4.8 DownCarrier

```
# First value:
value = repcap.DownCarrier.Dc1
# Range:
Dc1 .. Dc32
# All values (32x):
Dc1 | Dc2 | Dc3 | Dc4 | Dc5 | Dc6 | Dc7 | Dc8
Dc9 | Dc10 | Dc11 | Dc12 | Dc13 | Dc14 | Dc15 | Dc16
Dc17 | Dc18 | Dc19 | Dc20 | Dc21 | Dc22 | Dc23 | Dc24
Dc25 | Dc26 | Dc27 | Dc28 | Dc29 | Dc30 | Dc31 | Dc32
```

4.9 HSSCch

```
# First value:
value = repcap.HSSCch.No1
# Values (4x):
No1 | No2 | No3 | No4
```

4.10 IPversion

```
# First value:
value = repcap.IPversion.IPv4
# Values (2x):
IPv4 | IPv6
```

4.11 NonContigCell

```
# First value:
value = repcap.NonContigCell.Nc2
# Values (3x):
Nc2 | Nc3 | Nc4
```

4.12 PacketData

```
# First value:
value = repcap.PacketData.Pd8
# Range:
Pd8 .. Pd384
# All values (6x):
Pd8 | Pd16 | Pd32 | Pd64 | Pd128 | Pd384
```

4.13 QuadratureAM

```
# First value:
value = repcap.QuadratureAM.QAM16
# Values (2x):
QAM16 | QAM64
```

4.14 RefMeasChannel

```
# First value:
value = repcap.RefMeasChannel.Ch1
# Range:
Ch1 .. Ch5
# All values (5x):
Ch1 | Ch2 | Ch3 | Ch4 | Ch5
```

4.15 SecondCode

```
# First value:
value = repcap.SecondCode.Sc1
# Values (4x):
Sc1 | Sc2 | Sc3 | Sc4
```

4.16 Timer

```
# First value:
value = repcap.Timer.T313
# Values (4x):
T313 | T323 | T3212 | T3312
```

4.17 TransportBlock

```
# First value:
value = repcap.TransportBlock.TBl1
# Values (4x):
TBl1 | TBl2 | TBl3 | TBl4
```

4.18 TransTimeInterval

```
# First value:
value = repcap.TransTimeInterval.Tti2
# Values (2x):
Tti2 | Tti10
```

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

RSCMWWCDMASIG API STRUCTURE

Global RepCaps

```
driver = RsCmwWcdmaSig('TCPIP::192.168.2.101::HISLIP')
# Carrier range: C1 .. C32
rc = driver.repcap_carrier_get()
driver.repcap_carrier_set(repcap.Carrier.C1)
```

[4SP]# Instance range: Inst1 .. Inst16 rc = driver.repcap_instance_get() driver.repcap_instance_set(repcap.Instance.Inst1)

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

730 total commands, 19 Sub-groups, 0 group commands

Initializes new RsCmwWcdmaSig 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 RsCmwWcdmaSig session.

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

Creates a new RsCmwWcdmaSig 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:WCDMa:SIGNaling<Instance>:ETOE
CONFigure:WCDMa:SIGNaling<Instance>:ESCode
```

class Configure

Configure commands group definition. 464 total commands, 24 Sub-groups, 2 group commands

get_es_code() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ESCode
value: bool = driver.configure.get_es_code()
```

Enables audio tests involving the 'audio measurements' application in remote operation only. It can only be set in the signal OFF state.

return enable: OFF | ON

get_etoe() → bool

```
# SCPI: CONFigure:WCDMa: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) . DAU is required for IP-based data tests.

return end_to_end_enable: OFF | ON

set_es_code(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ESCode
driver.configure.set_es_code(enable = False)
```

Enables audio tests involving the 'audio measurements' application in remote operation only. It can only be set in the signal OFF state.

param enable OFF | ON

set_etoe(*end_to_end_enable: bool*) → None

```
# SCPI: CONFigure:WCDMa: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) . DAU is required for IP-based data tests.

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 Psettings

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:PSETtings:ERGM
CONFigure:WCDMa:SIGNaling<Instance>:PSETtings:HUMP
CONFigure:WCDMa:SIGNaling<Instance>:PSETtings
```

class Psettings

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

class ErgmStruct

Structure for reading output parameters. Fields:

- Test_Mode: enums.TestMode: HOLD | UPDown 'Missed Hold', 'Missed Up/Down'
- Tti: enums.TransTimeInterval: M2 | M10 2 ms, 10 ms

get_ergm() → ErgmStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:PSETtings:ERGM
value: ErgmStruct = driver.configure.psettings.get_ergm()
```

Selects mode and TTI for the 'E-RGCH Measurement'wizard.

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

get_hump() → RsCmwWcdmaSig.enums.SubTest

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:PSETtings:HUMP
value: enums.SubTest = driver.configure.psettings.get_hump()
```

Selects a subtest for the HSUPA maximum output power wizard.

return sub_test: S1 | S2 | S3 | S4 | S5 Subtest 1 to subtest 5

set_ergm(*value: RsCmwWcdmaSig.Implementations.Configure_.Psettings.Psettings.ErgmStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:PSETtings:ERGM
driver.configure.psettings.set_ergm(value = ErgmStruct())
```

Selects mode and TTI for the 'E-RGCH Measurement'wizard.

param value see the help for ErgmStruct structure arguments.

set_hump(*sub_test: RsCmwWcdmaSig.enums.SubTest*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:PSETtings:HUMP
driver.configure.psettings.set_hump(sub_test = enums.SubTest.S1)
```

Selects a subtest for the HSUPA maximum output power wizard.

param sub_test S1 | S2 | S3 | S4 | S5 Subtest 1 to subtest 5

set_value(*selection: RsCmwWcdmaSig.enums.WizzardSelection*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:PSETtings
driver.configure.psettings.set_value(selection = enums.WizzardSelection.DHIP)
```

Executes the wizard to apply the selected predefined set of WCDMA settings. INTRO_CMD_HELP: Configure the following selections before executing the wizard:

- 'General Settings'
- HUMP: see method RsCmwWcdmaSig.Configure.Psettings.hump
- ERGM: see method RsCmwWcdmaSig.Configure.Psettings.ergm

param selection HDMT | HUMT | HSMT | HUMP | DHIP | ERGM | HCQI | OOS HDMT: HSDPA maximum throughput HUMT: HSUPA maximum throughput HSMT: HSPA maximum throughput HUMP: HSUPA maximum output power DHIP: Dual carrier HSPA inner loop power control ERGM: HSUPA E-RGCH measurement HCQI: HSDPA CQI measurement OOS: Out-of-sync handling

7.1.2 Mmonitor

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:MMONitor:ENABle
```

class Mmonitor

Mmonitor commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:MMONitor:ENABle
value: bool = driver.configure.mmonitor.get_enable()
```

Enables or disables message monitoring for the WCDMA signaling application.

return enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:MMONitor:ENABle
driver.configure.mmonitor.set_enable(enable = False)
```

Enables or disables message monitoring for the WCDMA signaling application.

param enable OFF | ON

Cloning the Group

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

Subgroups

7.1.2.1 IpAddress

SCPI Commands

```
CONFigure:WCDMa: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.IpAddrIndex: IP1 | IP2 | IP3 Address pool index
- Ip_Address: str: Used IP address as string

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:MMONitor:IPADdress
value: GetStruct = driver.configure.mmonitor.ipAddress.get()
```

Selects the IP address to which signaling messages have to be sent for message monitoring. The address pool is configured globally via CONFigure:BASE:MMONitor:IPADdress<n>. A query returns both the current index and the resulting IP address.

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

set(*index: RsCmwWcdmaSig.enums.IpAddrIndex*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:MMONitor:IPADdress
driver.configure.mmonitor.ipAddress.set(index = enums.IpAddrIndex.IP1)
```

Selects the IP address to which signaling messages have to be sent for message monitoring. The address pool is configured globally via CONFigure:BASE:MMONitor:IPADdress<n>. A query returns both the current index and the resulting IP address.

param index IP1 | IP2 | IP3 Address pool index

7.1.3 UeReport

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:RINTerval
```

class UeReport

UeReport commands group definition. 7 total commands, 2 Sub-groups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:ENABle
value: bool = driver.configure.ueReport.get_enable()
```

Enables or disables the UE measurement report completely.

return enable: OFF | ON

get_rinterval() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:RINTerval
value: float = driver.configure.ueReport.get_rinterval()
```

Sets the interval between two consecutive measurement report messages.

return interval: Range: 0.25 s to 64 s, Unit: s

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:ENABle
driver.configure.ueReport.set_enable(enable = False)
```

Enables or disables the UE measurement report completely.

param enable OFF | ON

set_rinterval(*interval: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:RINTerval
driver.configure.ueReport.set_rinterval(interval = 1.0)
```

Sets the interval between two consecutive measurement report messages.

param interval Range: 0.25 s to 64 s, Unit: s

Cloning the Group

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

Subgroups

7.1.3.1 Ccell

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:CCELl:ENABle
```

class Ccell

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

class EnableStruct

Structure for reading output parameters. Fields:

- Cpi_Ch_Rscp: bool: OFF | ON
- Cpi_Ch_Ec_Io: bool: OFF | ON
- Tch_Bler: bool: OFF | ON
- Tx_Power: bool: OFF | ON
- Rx_Tx_Time_Diff: bool: OFF | ON
- Pathloss: bool: OFF | ON

get_enable() → EnableStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:CCELl:ENABle
value: EnableStruct = driver.configure.ueReport.ccell.get_enable()
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message for the current cell.

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

set_enable(*value: RsCmwWcdmaSig.Implementations.Configure_.UeReport_.Ccell.Ccell.EnableStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:CCELl:ENABle
driver.configure.ueReport.ccell.set_enable(value = EnableStruct())
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message for the current cell.

param value see the help for EnableStruct structure arguments.

7.1.3.2 Ncell

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:NCELl:ENABle
```

class Ncell

Ncell commands group definition. 4 total commands, 3 Sub-groups, 1 group commands

class EnableStruct

Structure for reading output parameters. Fields:

- Cpi_Ch_Rscp: bool: OFF | ON
- Cpi_Ch_Ec_Io: bool: OFF | ON
- Rssi: bool: OFF | ON
- Sfn_Cfn_Time_Diff: bool: OFF | ON
- Pathloss: bool: OFF | ON

get_enable() → EnableStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:ENABle
value: EnableStruct = driver.configure.ueReport.ncell.get_enable()
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message for carrier 2.

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

set_enable(*value: RsCmwWcdmaSig.Implementations.Configure_.UeReport_.Ncell.Ncell.EnableStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:ENABle
driver.configure.ueReport.ncell.set_enable(value = EnableStruct())
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message for carrier 2.

param value see the help for EnableStruct structure arguments.

Cloning the Group

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

Subgroups

7.1.3.2.1 Gsm

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:NCELl:GSM:ENABle
```

class Gsm

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

class EnableStruct

Structure for reading output parameters. Fields:

- Rssi: bool: OFF | ON
- Bsic: bool: OFF | ON

get_enable() → EnableStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:GSM:ENABle
value: EnableStruct = driver.configure.ueReport.ncell.gsm.get_enable()
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message related to GSM neighbor cell. BSIC measurement requires activated RSSI measurement.

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

set_enable(*value: RsCmwWcdmaSig.Implementations.Configure_.UeReport_.Ncell_.Gsm.Gsm.EnableStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:GSM:ENABle
driver.configure.ueReport.ncell.gsm.set_enable(value = EnableStruct())
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message related to GSM neighbor cell. BSIC measurement requires activated RSSI measurement.

param value see the help for EnableStruct structure arguments.

7.1.3.2.2 Wcdma

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:NCELl:WCDMa:ENABle
```

class Wcdma

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

class EnableStruct

Structure for reading output parameters. Fields:

- Rscp: bool: OFF | ON

- Ecn_0: bool: OFF | ON
- Rssi: bool: OFF | ON
- Sfn_Cfn: bool: OFF | ON
- Pathloss: bool: OFF | ON

get_enable() → EnableStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:WCDMa:ENABle
value: EnableStruct = driver.configure.ueReport.ncell.wcdma.get_enable()
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message related to WCDMA neighbor cell.

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

set_enable(*value: RsCmwWcdmaSig.Implementations.Configure_.UeReport_.Ncell_.Wcdma.Wcdma.EnableStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:WCDMa:ENABle
driver.configure.ueReport.ncell.wcdma.set_enable(value = EnableStruct())
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message related to WCDMA neighbor cell.

param value see the help for EnableStruct structure arguments.

7.1.3.2.3 Lte

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:NCELl:LTE:ENABle
```

class Lte

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

class EnableStruct

Structure for reading output parameters. Fields:

- Rsrp: bool: OFF | ON
- Rsrq: bool: OFF | ON

get_enable() → EnableStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:LTE:ENABle
value: EnableStruct = driver.configure.ueReport.ncell.lte.get_enable()
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message related to LTE neighbor cell.

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

set_enable(*value: RsCmwWcdmaSig.Implementations.Configure_.UeReport_.Ncell_.Lte.Lte.EnableStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UEReport:NCELl:LTE:ENABle
driver.configure.ueReport.ncell.lte.set_enable(value = EnableStruct())
```

Enables or disables the evaluation and display of the individual information elements included in the UE measurement report message related to LTE neighbor cell.

param value see the help for EnableStruct structure arguments.

7.1.4 Cmode

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CMODe:PATTern
```

class Cmode

Cmode commands group definition. 7 total commands, 3 Sub-groups, 1 group commands

get_pattern() → RsCmwWcdmaSig.enums.CmodePatternSelection

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:PATTern
value: enums.CmodePatternSelection = driver.configure.cmode.get_pattern()
```

Selects the transmission gap patterns for compressed mode.

return selection: NONE | UEReport | SINGle | ULCM NONE: compressed mode disabled UEReport: several patterns for different measurement purposes used in parallel See method RsCmwWcdmaSig.Configure.Cmode.UeReport.enable SINGle: selectable pattern for a definite measurement purpose See method RsCmwWcdmaSig.Configure.Cmode.Single.typePy ULCM: selectable pattern for the UL compressed mode TX test See method RsCmwWcdmaSig.Configure.Cmode.Ulcm.typePy

set_pattern(*selection: RsCmwWcdmaSig.enums.CmodePatternSelection*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:PATTern
driver.configure.cmode.set_pattern(selection = enums.CmodePatternSelection.NONE)
```

Selects the transmission gap patterns for compressed mode.

param selection NONE | UEReport | SINGle | ULCM NONE: compressed mode disabled UEReport: several patterns for different measurement purposes used in parallel See method RsCmwWcdmaSig.Configure.Cmode.UeReport.enable SINGle: selectable pattern for a definite measurement purpose See method RsCmwWcdmaSig.Configure.Cmode.Single.typePy ULCM: selectable pattern for the UL compressed mode TX test See method RsCmwWcdmaSig.Configure.Cmode.Ulcm.typePy

Cloning the Group

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

Subgroups

7.1.4.1 Ulcm

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CMODe:ULCM:TYPE
```

class Ulcm

Ulcm commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get_type_py() → RsCmwWcdmaSig.enums.TransGapType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:ULCM:TYPE
value: enums.TransGapType = driver.configure.cmode.ulcm.get_type_py()
```

Selects the transmission gap patterns for the UL compressed mode TX test.

return type_py: AR | AF | B AR: pattern A (rising TPC) defined in 3GPP TS 34.121, table 5.7.6 AF: pattern A (falling TPC) defined in 3GPP TS 34.121, table 5.7.7 B: pattern B defined in 3GPP TS 34.121, table 5.7.8

set_type_py(*type_py: RsCmwWcdmaSig.enums.TransGapType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:ULCM:TYPE
driver.configure.cmode.ulcm.set_type_py(type_py = enums.TransGapType.AF)
```

Selects the transmission gap patterns for the UL compressed mode TX test.

param type_py AR | AF | B AR: pattern A (rising TPC) defined in 3GPP TS 34.121, table 5.7.6 AF: pattern A (falling TPC) defined in 3GPP TS 34.121, table 5.7.7 B: pattern B defined in 3GPP TS 34.121, table 5.7.8

Cloning the Group

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

Subgroups

7.1.4.1.1 Activation

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CMODe:ULCM:ACTivation
```

class Activation

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:ULCM:ACTivation
driver.configure.cmode.ulcm.activation.set()
```

Activates the selected pattern type for the UL compressed mode TX test.

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:ULCM:ACTivation
driver.configure.cmode.ulcm.activation.set_with_opc()
```

Activates the selected pattern type for the UL compressed mode TX test.

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

7.1.4.2 Single

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CMODe:SINGle:TYPE
CONFigure:WCDMa:SIGNaling<Instance>:CMODe:SINGle:ACTivation
```

class Single

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

get_activation() → RsCmwWcdmaSig.enums.CmodeActivation

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:SINGle:ACTivation
value: enums.CmodeActivation = driver.configure.cmode.single.get_activation()
```

Selects whether the compressed mode has to be activated for the whole duration of the connection (RAB setup) or for the duration of a UE report measurement only.

return activation: RAB | MEASurement

get_type_py() → RsCmwWcdmaSig.enums.TransGapTypeExtended

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:SINGle:TYPE
value: enums.TransGapTypeExtended = driver.configure.cmode.single.get_type_py()
```

Selects the single transmission gap patterns for a definite measurement purpose.

return type_py: RFA | RFB | A | B | C | D | E | F RFA: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table 5.7.5) RFB: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table 5.7.8) A: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table C.5.2, set 1) B: for GSM neighbor cell measurements (see 3GPP TS 34.121, table C.5.2, set 2) C: to search for the BSIC and decode it (see 3GPP TS 25.133, table 8.7, pattern 2) D: to track and decode the BSIC after an initial BSIC identification (see 3GPP TS 25.133, table 8.8, pattern 2) E: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table C.5.1 set 1) F:

set_activation(*activation: RsCmwWcdmaSig.enums.CmodeActivation*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:SINGle:ACTivation
driver.configure.cmode.single.set_activation(activation = enums.CmodeActivation.
↪MEASurement)
```

Selects whether the compressed mode has to be activated for the whole duration of the connection (RAB setup) or for the duration of a UE report measurement only.

param activation RAB | MEASurement

set_type_py(*type_py: RsCmwWcdmaSig.enums.TransGapTypeExtended*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:SINGle:TYPE
driver.configure.cmode.single.set_type_py(type_py = enums.TransGapTypeExtended.
↪A)
```

Selects the single transmission gap patterns for a definite measurement purpose.

param type_py RFA | RFB | A | B | C | D | E | F RFA: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table 5.7.5) RFB: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table 5.7.8) A: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table C.5.2, set 1) B: for GSM neighbor cell measurements (see 3GPP TS 34.121, table C.5.2, set 2) C: to search for the BSIC and decode it (see 3GPP TS 25.133, table 8.7, pattern 2) D: to track and decode the BSIC after an initial BSIC identification (see 3GPP TS 25.133, table 8.8, pattern 2) E: for WCDMA neighbor cell measurements (see 3GPP TS 34.121, table C.5.1 set 1) F:

7.1.4.3 UeReport

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CMODe:UEReport:ACTivation
CONFigure:WCDMa:SIGNaling<Instance>:CMODe:UEReport:ENABle
```

class UeReport

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

class ActivationStruct

Structure for reading output parameters. Fields:

- Fdd: enums.CmodeActivation: RAB | MEASurement
- Gsm_Rssi: enums.CmodeActivation: RAB | MEASurement

- Gsm_Bsic: enums.CmodeActivation: RAB | MEASurement
- Gsm_Bsic_Reconf: enums.CmodeActivation: RAB | MEASurement
- Eutra: enums.CmodeActivation: RAB | MEASurement

class EnableStruct

Structure for reading output parameters. Fields:

- Fdd: bool: OFF | ON
- Gsm_Rssi: bool: OFF | ON
- Gsm_Bsic: bool: OFF | ON
- Gsm_Bsic_Reconf: bool: OFF | ON
- Eutra: bool: OFF | ON

get_activation() → ActivationStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:UEReport:ACTivation
value: ActivationStruct = driver.configure.cmode.ueReport.get_activation()
```

Selects whether the compressed mode pattern has to be activated for the whole duration of the connection (RAB setup) or for the duration of a specified UE report measurement only.

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

get_enable() → EnableStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:UEReport:ENABle
value: EnableStruct = driver.configure.cmode.ueReport.get_enable()
```

Enables the transmission gap patterns for different measurement purposes. All selected patterns are used in parallel.

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

set_activation(*value: RsCmwWcdmaSig.Implementations.Configure_.Cmode_.UeReport.UeReport.ActivationStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:UEReport:ACTivation
driver.configure.cmode.ueReport.set_activation(value = ActivationStruct())
```

Selects whether the compressed mode pattern has to be activated for the whole duration of the connection (RAB setup) or for the duration of a specified UE report measurement only.

param value see the help for ActivationStruct structure arguments.

set_enable(*value: RsCmwWcdmaSig.Implementations.Configure_.Cmode_.UeReport.UeReport.EnableStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CMODe:UEReport:ENABle
driver.configure.cmode.ueReport.set_enable(value = EnableStruct())
```

Enables the transmission gap patterns for different measurement purposes. All selected patterns are used in parallel.

param value see the help for EnableStruct structure arguments.

7.1.5 RfSettings

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:DBDC
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:ENPMode
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:ENPower
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:MARGin
```

class RfSettings

RfSettings commands group definition. 32 total commands, 5 Sub-groups, 4 group commands

class DbdcStruct

Structure for reading output parameters. Fields:

- Enable: bool: OFF | ON
- Config: enums.OperBandConfig: UDEFined | C1 | C2 | C3 | C4 | C5 | C6 UDEFined: User defined (custom) - free band selection C1: DL band A I, DL band B VIII C2: DL band A II, DL band B IV C3: DL band A I, DL band B V C4: DL band A I, DL band B XI C5: DL band A II, DL band B V C6: DL band A I, DL band B XXXII UL applies the band of the DL carrier 1, where the assignment of band A or band B is possible. Exception: no UL for operating band XXXII.

get_dbdc() → DbdcStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:DBDC
value: DbdcStruct = driver.configure.rfSettings.get_dbdc()
```

Enables dual band dual carrier HSDPA operation and selects the operating bands for UL and DL. For operating band description, see 'Operating Bands'.

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

get_enp_mode() → RsCmwWcdmaSig.enums.NominalPowerMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:ENPMode
value: enums.NominalPowerMode = driver.configure.rfSettings.get_enp_mode()
```

Selects the expected nominal power mode. The expected nominal power of the UL signal can be defined manually or calculated automatically, according to the UL power control settings.

INTRO_CMD_HELP: For manual configuration, see:

- method RsCmwWcdmaSig.Configure.RfSettings.envelopePower
- method RsCmwWcdmaSig.Configure.RfSettings.margin

return mode: MANual | ULPC MANual: The expected nominal power and margin are specified manually. ULPC: The expected nominal power is calculated according to the UL power control settings.

get_envelope_power() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:ENPower
value: float = driver.configure.rfSettings.get_envelope_power()
```

Sets the expected nominal power of the measured RF signal.

return expected_power: The range of the expected nominal power can be calculated as follows: Range (Expected Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the data sheet. Unit: dBm

get_margin() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:MARGin
value: float = driver.configure.rfSettings.get_margin()
```

Sets the margin that the R&S CMW adds to the expected nominal power to determine the reference level in manual mode. The reference level minus the external input attenuation must be within the power range of the selected input connector; refer to the data sheet.

INTRO_CMD_HELP: Refer also to the following commands:

- method RsCmwWcdmaSig.Configure.RfSettings.enpMode
- method RsCmwWcdmaSig.Configure.RfSettings.envelopePower
- method RsCmwWcdmaSig.Configure.RfSettings.Carrier.Eattenuation.inputPy

return user_margin: Range: 0 dB to (34 dB + external attenuation - expected nominal power) , Unit: dB

set_dbdc(*value: RsCmwWcdmaSig.Implementations.Configure_.RfSettings.RfSettings.DbdcStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:DBDC
driver.configure.rfSettings.set_dbdc(value = DbdcStruct())
```

Enables dual band dual carrier HSDPA operation and selects the operating bands for UL and DL. For operating band description, see 'Operating Bands'.

param value see the help for DbdcStruct structure arguments.

set_enp_mode(*mode: RsCmwWcdmaSig.enums.NominalPowerMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:ENPMode
driver.configure.rfSettings.set_enp_mode(mode = enums.NominalPowerMode.
↪AUToranging)
```

Selects the expected nominal power mode. The expected nominal power of the UL signal can be defined manually or calculated automatically, according to the UL power control settings.

INTRO_CMD_HELP: For manual configuration, see:

- method RsCmwWcdmaSig.Configure.RfSettings.envelopePower
- method RsCmwWcdmaSig.Configure.RfSettings.margin

param mode MANual | ULPC MANual: The expected nominal power and margin are specified manually. ULPC: The expected nominal power is calculated according to the UL power control settings.

set_envelope_power(*expected_power: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:ENPower
driver.configure.rfSettings.set_envelope_power(expected_power = 1.0)
```

Sets the expected nominal power of the measured RF signal.

param expected_power The range of the expected nominal power can be calculated as follows: Range (Expected Power) = Range (Input Power) + External Attenuation - User Margin The input power range is stated in the data sheet. Unit: dBm

set_margin(*user_margin: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:MARGin
driver.configure.rfSettings.set_margin(user_margin = 1.0)
```

Sets the margin that the R&S CMW adds to the expected nominal power to determine the reference level in manual mode. The reference level minus the external input attenuation must be within the power range of the selected input connector; refer to the data sheet.

INTRO_CMD_HELP: Refer also to the following commands:

- method RsCmwWcdmaSig.Configure.RfSettings.enpMode
- method RsCmwWcdmaSig.Configure.RfSettings.envelopePower
- method RsCmwWcdmaSig.Configure.RfSettings.Carrier.Eattenuation.inputPy

param user_margin Range: 0 dB to (34 dB + external attenuation - expected nominal power) , Unit: dB

Cloning the Group

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

Subgroups

7.1.5.1 Dcarrier

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:DCARrier:SEParation
```

class Dcarrier

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

get_separation() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:DCARrier:SEParation
value: float = driver.configure.rfSettings.dcarrier.get_separation()
```

Sets the frequency separation for particular carrier frequencies in multi-carrier scenario.

return dc_freq_sep: Range: 0 MHz to 10 MHz , Unit: Hz

set_separation(*dc_freq_sep: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:DCARrier:SEParation
driver.configure.rfSettings.dcarrier.set_separation(dc_freq_sep = 1.0)
```

Sets the frequency separation for particular carrier frequencies in multi-carrier scenario.

param dc_freq_sep Range: 0 MHz to 10 MHz , Unit: Hz

7.1.5.2 Carrier

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:GMTFactor
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:AWGN
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:COPower
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:TOPower
```

class Carrier

Carrier commands group definition. 16 total commands, 7 Sub-groups, 4 group commands

class AwgnStruct

Structure for reading output parameters. Fields:

- Enable: bool: OFF | ON Enables or disables the AWGN signal
- Level: float: The range of the AWGN level can be calculated as follows from the range of the output power stated below: Min (AWGN) = Min (Output Power) - External Attenuation Max (AWGN) = Max (Output Power) - External Attenuation - Base Level Range: -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 , Unit: dBm

get_awgn() → AwgnStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:AWGN
value: AwgnStruct = driver.configure.rfSettings.carrier.get_awgn()
```

Enables or disables AWGN insertion via the signaling unit and sets the total AWGN level within the channel bandwidth. For multi-carrier, the same settings are applied to all carriers. Thus it is sufficient to configure one carrier.

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

Global Repeated Capabilities: repcap.Carrier

get_co_power() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:COPower
value: float = driver.configure.rfSettings.carrier.get_co_power()
```

Sets the base level of the generator. For multi-carrier, it can be set per carrier. The allowed value range can be calculated as follows: Range (Base Level) = Range (Output Power) - External Attenuation - Insertion Loss + Baseband Level 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. Insertion loss is only relevant for internal fading. Baseband level only relevant for external fading.

return out_channel_pow: Range: see above , Unit: dBm

Global Repeated Capabilities: repcap.Carrier

get_gmt_factor() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:GMTFactor
value: float = driver.configure.rfSettings.carrier.get_gmt_factor()
```

Queries the ratio of the output channel power (Ior) to the AWGN power (Ioc) . INV indicates that AWGN noise is disabled.

return ratio: Range: -25.4 dB to 44.9 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

get_to_power() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:TOPower
value: float = driver.configure.rfSettings.carrier.get_to_power()
```

Queries the sum of the output channel power (Ior) and the AWGN power (Ioc) .

return total_output_pow: Unit: dBm

Global Repeated Capabilities: repcap.Carrier

set_awgn(*value: RsCmwWcdmaSig.Implementations.Configure_.RfSettings_.Carrier.Carrier.AwgnStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:AWGN
driver.configure.rfSettings.carrier.set_awgn(value = AwgnStruct())
```

Enables or disables AWGN insertion via the signaling unit and sets the total AWGN level within the channel bandwidth. For multi-carrier, the same settings are applied to all carriers. Thus it is sufficient to configure one carrier.

param value see the help for AwgnStruct structure arguments.

Global Repeated Capabilities: repcap.Carrier

set_co_power(*out_channel_pow: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:COPower
driver.configure.rfSettings.carrier.set_co_power(out_channel_pow = 1.0)
```

Sets the base level of the generator. For multi-carrier, it can be set per carrier. The allowed value range can be calculated as follows: Range (Base Level) = Range (Output Power) - External Attenuation - Insertion Loss + Baseband Level 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. Insertion loss is only relevant for internal fading. Baseband level only relevant for external fading.

param out_channel_pow Range: see above , Unit: dBm

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.5.2.1 Edc

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:EDC:INPut
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:EDC:OUTPut
```

class Edc

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

get_input_py() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EDC:INPut
value: float = driver.configure.rfSettings.carrier.edc.get_input_py()
```

No command help available

return ext_delay: Range: 0 s to 20E-6 s

Global Repeated Capabilities: repcap.Carrier

get_output() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EDC:OUTPut
value: float = driver.configure.rfSettings.carrier.edc.get_output()
```

No command help available

return ext_delay: Range: 0 s to 20E-6 s

Global Repeated Capabilities: repcap.Carrier

set_input_py(*ext_delay: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EDC:INPut
driver.configure.rfSettings.carrier.edc.set_input_py(ext_delay = 1.0)
```

No command help available

param ext_delay Range: 0 s to 20E-6 s

Global Repeated Capabilities: repcap.Carrier

set_output(*ext_delay: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EDC:OUTPut
driver.configure.rfSettings.carrier.edc.set_output(ext_delay = 1.0)
```

No command help available

param ext_delay Range: 0 s to 20E-6 s

Global Repeated Capabilities: repcap.Carrier

7.1.5.2.2 Eattenuation

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:EATTenuation:INPut
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:EATTenuation:OUTPut
```

class Eattenuation

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

get_input_py() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EATTenuation:INPut
value: float = driver.configure.rfSettings.carrier.eattenuation.get_input_py()
```

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

return ext_attenuation: Range: -50 dB to 90 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

get_output() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EATTenuation:OUTPut
value: float = driver.configure.rfSettings.carrier.eattenuation.get_output()
```

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

return ext_attenuation: Range: -50 dB to 90 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

set_input_py(*ext_attenuation: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EATTenuation:INPut
driver.configure.rfSettings.carrier.eattenuation.set_input_py(ext_attenuation =␣
↪1.0)
```

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

param ext_attenuation Range: -50 dB to 90 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

set_output(*ext_attenuation: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:EATTenuation:OUTPut
driver.configure.rfSettings.carrier.eattenuation.set_output(ext_attenuation = 1.
↪0)
```

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

param ext_attenuation Range: -50 dB to 90 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

7.1.5.2.3 Channel

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:CHANnel:UL
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:CHANnel:DL
```

class Channel

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

get_downlink() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:CHANnel:DL
value: int = driver.configure.rfSettings.carrier.channel.get_downlink()
```

Selects the DL channel number. The channel number must be valid for the current operating band, for dependencies see 'Operating Bands'. The related UL channel number is calculated and set automatically. For multi-carrier scenarios, the channel numbers of the other carriers are calculated and set as well.

return channel_number: Range: depends on operating band

Global Repeated Capabilities: repcap.Carrier

get_uplink() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:CHANnel:UL
value: int = driver.configure.rfSettings.carrier.channel.get_uplink()
```

Selects the UL channel number. The channel number must be valid for the current operating band. For dependencies, see 'Operating Bands'. The related DL channel number is calculated and set automatically.

return channel_number: Range: depends on operating band

Global Repeated Capabilities: repcap.Carrier

set_downlink(*channel_number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:CHANnel:DL
driver.configure.rfSettings.carrier.channel.set_downlink(channel_number = 1)
```

Selects the DL channel number. The channel number must be valid for the current operating band, for dependencies see 'Operating Bands'. The related UL channel number is calculated and set automatically. For multi-carrier scenarios, the channel numbers of the other carriers are calculated and set as well.

param channel_number Range: depends on operating band

Global Repeated Capabilities: repcap.Carrier

set_uplink(*channel_number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:CHANnel:UL
driver.configure.rfSettings.carrier.channel.set_uplink(channel_number = 1)
```

Selects the UL channel number. The channel number must be valid for the current operating band. For dependencies, see 'Operating Bands'. The related DL channel number is calculated and set automatically.

param channel_number Range: depends on operating band

Global Repeated Capabilities: repcap.Carrier

7.1.5.2.4 Frequency

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:FREQuency:UL
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:FREQuency:DL
```

class Frequency

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

get_downlink() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FREQuency:DL
value: float = driver.configure.rfSettings.carrier.frequency.get_downlink()
```

Selects the DL carrier center frequency. The frequency must correspond to a channel valid for the current operating band, for dependencies see 'Operating Bands'. The related UL frequency is calculated and set automatically. For dual carrier, the frequency of the other carrier is calculated and set as well.

return frequency: Range: depends on operating band , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

get_uplink() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FREQuency:UL
value: float = driver.configure.rfSettings.carrier.frequency.get_uplink()
```

Selects the UL carrier center frequency. The frequency must correspond to a channel valid for the current operating band. For dependencies, see 'Operating Bands'. The related DL frequency is calculated and set automatically.

return frequency: Range: depends on operating band , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

set_downlink(*frequency: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FREQuency:DL
driver.configure.rfSettings.carrier.frequency.set_downlink(frequency = 1.0)
```

Selects the DL carrier center frequency. The frequency must correspond to a channel valid for the current operating band, for dependencies see 'Operating Bands'. The related UL frequency is calculated and set automatically. For dual carrier, the frequency of the other carrier is calculated and set as well.

param frequency Range: depends on operating band , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

set_uplink(*frequency: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FREQuency:UL
driver.configure.rfSettings.carrier.frequency.set_uplink(frequency = 1.0)
```

Selects the UL carrier center frequency. The frequency must correspond to a channel valid for the current operating band. For dependencies, see 'Operating Bands'. The related DL frequency is calculated and set automatically.

param frequency Range: depends on operating band , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

7.1.5.2.5 Foffset

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:FOFFset:UL
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:FOFFset:DL
```

class Foffset

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

get_downlink() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FOFFset:DL
value: float = driver.configure.rfSettings.carrier.foffset.get_downlink()
```

Specifies a positive or negative frequency offset to be added to the downlink center frequency of the configured channel, see method RsCmwWcdmaSig.Configure.RfSettings.Carrier.Frequency.downlink.

return freq_offset: Range: -100000 Hz to 100000 Hz , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

get_uplink() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FOFFset:UL
value: float = driver.configure.rfSettings.carrier.foffset.get_uplink()
```

Specifies a positive or negative frequency offset to be added to the uplink center frequency of the configured channel, see method RsCmwWcdmaSig.Configure.RfSettings.Carrier.Frequency.uplink .

return freq_offset: Range: -100000 Hz to 100000 Hz , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

set_downlink(*freq_offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FOFFset:DL
driver.configure.rfSettings.carrier.foffset.set_downlink(freq_offset = 1.0)
```

Specifies a positive or negative frequency offset to be added to the downlink center frequency of the configured channel, see method RsCmwWcdmaSig.Configure.RfSettings.Carrier.Frequency.downlink.

param freq_offset Range: -100000 Hz to 100000 Hz , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

set_uplink(*freq_offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>
↪:FOFFset:UL
driver.configure.rfSettings.carrier.foffset.set_uplink(freq_offset = 1.0)
```

Specifies a positive or negative frequency offset to be added to the uplink center frequency of the configured channel, see method RsCmwWcdmaSig.Configure.RfSettings.Carrier.Frequency.uplink .

param freq_offset Range: -100000 Hz to 100000 Hz , Unit: Hz

Global Repeated Capabilities: repcap.Carrier

7.1.5.2.6 Uplink

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:UL
```

class Uplink

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

class GetStruct

Response structure. Fields:

- Band: enums.OperationBand: No parameter help available
- Channel: int: No parameter help available
- Frequency: float: No parameter help available

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:UL
value: GetStruct = driver.configure.rfSettings.carrier.uplink.get()
```

No command help available

Global Repeated Capabilities: repcap.Carrier

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

set(*band: RsCmwWcdmaSig.enums.OperationBand*, *channel: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:UL
driver.configure.rfSettings.carrier.uplink.set(band = enums.OperationBand.OB1,␣
↪channel = 1)
```

No command help available

param band No help available

param channel No help available

Global Repeated Capabilities: repcap.Carrier

7.1.5.2.7 Downlink

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:CARRier<Carrier>:DL
```

class Downlink

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

class GetStruct

Response structure. Fields:

- Band: enums.OperationBand: OB1 | … | OB14 | OB19 | … | OB22 | OB25 | OB26 | OB32 | OBS1 | … | OBS3 | OBL1 | UDEFined OB1, …, OB14: operating band I to XIV OB19, …, OB22: operating band XIX to XXII OB25, OB26: operating band XXV, XXVI OB32: operating band XXXII (restricted to dual band scenarios) OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L UDEFined: user defined
- Channel: int: Range: depends on operating band
- Frequency: float: A query returns band, channel number and corresponding carrier center frequency Range: depends on operating band

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:DL
value: GetStruct = driver.configure.rfSettings.carrier.downlink.get()
```

Selects the operating band and the DL channel number. The channel number must be valid for the operating band, for dependencies see 'Operating Bands'. The related UL channel number is calculated and set automatically. For scenarios with multi-carrier, the channel numbers of the other carriers are calculated and set as well.

Global Repeated Capabilities: repcap.Carrier

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

set(*band: RsCmwWcdmaSig.enums.OperationBand*, *channel: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:CARRier<carrier>:DL
driver.configure.rfSettings.carrier.downlink.set(band = enums.OperationBand.OB1,
↪ channel = 1)
```

Selects the operating band and the DL channel number. The channel number must be valid for the operating band, for dependencies see 'Operating Bands'. The related UL channel number is calculated and set automatically. For scenarios with multi-carrier, the channel numbers of the other carriers are calculated and set as well.

param band OB1 | … | OB14 | OB19 | … | OB22 | OB25 | OB26 | OB32 | OBS1 | … | OBS3 | OBL1 | UDEFined OB1, …, OB14: operating band I to XIV OB19, …, OB22: operating band XIX to XXII OB25, OB26: operating band XXV, XXVI OB32: operating band XXXII (restricted to dual band scenarios) OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L UDEFined: user defined

param channel Range: depends on operating band

Global Repeated Capabilities: repcap.Carrier

7.1.5.3 CoPower

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:COPower:TOTal
```

class CoPower

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

get_total() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:COPower:TOTal
value: float = driver.configure.rfSettings.coPower.get_total()
```

Sets the total base level of the generator. For multi-carrier operation, this value is the sum of all carrier powers. If you modify the total power level, all carrier powers are increased/decreased by the same amount so that the new total power level is reached. The allowed value range per carrier can be calculated as follows: Range (Base Level) = Range (Output Power) - External Attenuation - Insertion Loss + Baseband Level Range (Output Power) = -130 dBm to -5 dBm (RFx COM) or -120 dBm to 3 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet. Insertion loss is only relevant for internal fading. Baseband level only relevant for external fading.

return total_out_ch_pwr: Range: see above , Unit: dBm

set_total(*total_out_ch_pwr: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:COPower:TOTal
driver.configure.rfSettings.coPower.set_total(total_out_ch_pwr = 1.0)
```

Sets the total base level of the generator. For multi-carrier operation, this value is the sum of all carrier powers. If you modify the total power level, all carrier powers are increased/decreased by the same amount so that the new total power level is reached. The allowed value range per carrier can be calculated as follows: Range (Base Level) = Range (Output Power) - External Attenuation - Insertion Loss + Baseband Level Range (Output Power) = -130 dBm to -5 dBm (RFx COM) or -120 dBm to 3 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet. Insertion loss is only relevant for internal fading. Baseband level only relevant for external fading.

param total_out_ch_pwr Range: see above , Unit: dBm

7.1.5.4 ToPower

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:TOPower:TOTal
```

class ToPower

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

get_total() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:TOPower:TOTal
value: float = driver.configure.rfSettings.toPower.get_total()
```

Queries the sum of the output channel power (Ior) and the AWGN power (Ioc) . For scenarios with multi-carrier, the result indicates the sum of the Ior and Ioc values of all carriers.

return comb_tot_out_pwr: Unit: dBm

7.1.5.5 UserDefined

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:UDSeparation
```

class UserDefined

UserDefined commands group definition. 9 total commands, 2 Sub-groups, 1 group commands

get_ud_separation() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:UDEFined:UDSeparation
value: float = driver.configure.rfSettings.userDefined.get_ud_separation()
```

Specifies the uplink - downlink separation interval for user-defined band.

return frequency: Range: -3832.4 Hz to 2012.4 Hz , Unit: Hz

set_ud_separation(*frequency: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:RFSettings:UDEFined:UDSeparation
driver.configure.rfSettings.userDefined.set_ud_separation(frequency = 1.0)
```

Specifies the uplink - downlink separation interval for user-defined band.

param frequency Range: -3832.4 Hz to 2012.4 Hz , Unit: Hz

Cloning the Group

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

Subgroups

7.1.5.5.1 Channel

class Channel

Channel 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.rfSettings.userDefined.channel.clone()
```

Subgroups

7.1.5.5.1.1 Downlink

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:CHANnel:DL:MINimum
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:CHANnel:DL:MAXimum
```

class Downlink

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

get_maximum() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:CHANnel:DL:MAXimum
value: int = driver.configure.rfSettings.userDefined.channel.downlink.get_
↪maximum()
```

Specifies the maximum value for downlink channel number within a user-defined band.

return channel: Range: 0 Ch to 16.383E+3 Ch

get_minimum() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:CHANnel:DL:MINimum
value: int = driver.configure.rfSettings.userDefined.channel.downlink.get_
↪minimum()
```

Specifies the minimum value for downlink channel number within a user-defined band.

return channel: Range: 0 Ch to 16.383E+3 Ch

set_maximum(*channel: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:CHANnel:DL:MAXimum
driver.configure.rfSettings.userDefined.channel.downlink.set_maximum(channel =␣
↪1)
```

Specifies the maximum value for downlink channel number within a user-defined band.

param channel Range: 0 Ch to 16.383E+3 Ch

set_minimum(*channel: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:CHANnel:DL:MINimum
driver.configure.rfSettings.userDefined.channel.downlink.set_minimum(channel =␣
↪1)
```

Specifies the minimum value for downlink channel number within a user-defined band.

param channel Range: 0 Ch to 16.383E+3 Ch

7.1.5.5.1.2 Uplink

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:CHANnel:UL:MINimum
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:CHANnel:UL:MAXimum
```

class Uplink

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

get_maximum() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:CHANnel:UL:MAXimum
value: int = driver.configure.rfSettings.userDefined.channel.uplink.get_
↪maximum()
```

Specifies the maximum value for uplink channel number within a user-defined band.

return channel: Range: 0 Ch to 16.383E+3 Ch

get_minimum() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:CHANnel:UL:MINimum
value: int = driver.configure.rfSettings.userDefined.channel.uplink.get_
↪minimum()
```

Specifies the minimum value for uplink channel number within a user-defined band.

return channel: Range: 0 Ch to 16.383E+3 Ch

set_minimum(*channel: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:CHANnel:UL:MINimum
driver.configure.rfSettings.userDefined.channel.uplink.set_minimum(channel = 1)
```

Specifies the minimum value for uplink channel number within a user-defined band.

param channel Range: 0 Ch to 16.383E+3 Ch

7.1.5.5.2 Frequency

class Frequency

Frequency 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.rfSettings.userDefined.frequency.clone()
```

Subgroups

7.1.5.5.2.1 Downlink

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:FREQuency:DL:MINimum
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:FREQuency:DL:MAXimum
```

class Downlink

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

get_maximum() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:FREQuency:DL:MAXimum
value: float = driver.configure.rfSettings.userDefined.frequency.downlink.get_
↪maximum()
```

Specifies the maximum value for downlink frequencies within a user-defined band.

return frequency: Range: 100 MHz to 6 GHz, Unit: Hz

get_minimum() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:FREQuency:DL:MINimum
value: float = driver.configure.rfSettings.userDefined.frequency.downlink.get_
↪minimum()
```

Specifies the minimum value for downlink frequencies within a user-defined band.

return frequency: Range: 100 MHz to 6 GHz, Unit: Hz

set_minimum(*frequency: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:FREQuency:DL:MINimum
driver.configure.rfSettings.userDefined.frequency.downlink.set_
↪minimum(frequency = 1.0)
```

Specifies the minimum value for downlink frequencies within a user-defined band.

param frequency Range: 100 MHz to 6 GHz, Unit: Hz

7.1.5.5.2.2 Uplink

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:FREQuency:UL:MINimum
CONFigure:WCDMa:SIGNaling<Instance>:RFSettings:UDEFined:FREQuency:UL:MAXimum
```

class Uplink

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

get_maximum() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:FREQuency:UL:MAXimum
value: float = driver.configure.rfSettings.userDefined.frequency.uplink.get_
↪maximum()
```

Specifies the maximum value for uplink frequencies within a user-defined band.

return frequency: Range: 100 MHz to 6 GHz, Unit: Hz

get_minimum() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:FREQuency:UL:MINimum
value: float = driver.configure.rfSettings.userDefined.frequency.uplink.get_
↪minimum()
```

Specifies the minimum value for uplink frequencies within a user-defined band.

return frequency: Range: 100 MHz to 6 GHz, Unit: Hz

set_minimum(*frequency: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:RFSettings:UDEFined:FREQuency:UL:MINimum
driver.configure.rfSettings.userDefined.frequency.uplink.set_minimum(frequency␣
↪= 1.0)
```

Specifies the minimum value for uplink frequencies within a user-defined band.

param frequency Range: 100 MHz to 6 GHz, Unit: Hz

7.1.6 Carrier

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CARRier<Carrier>:BAND
```

class Carrier

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

get_band() → RsCmwWcdmaSig.enums.OperationBand

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CARRier<carrier>:BAND
value: enums.OperationBand = driver.configure.carrier.get_band()
```

Selects the operating band (OB) . In single-band scenarios, all carriers use the same band. If you change it for one carrier, it is also changed for the other carriers.

return operation_band: OB1 | … | OB14 | OB19 | … | OB22 | OB25 | OB26 | OB32 | OBS1 | … | OBS3 | OBL1 | UDEFined OB1, …, OB14: operating band I to XIV OB19, …, OB22: operating band XIX to XXII OB25, OB26: operating band XXV, XXVI OB32: operating band XXXII (restricted to dual band scenarios) OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L UDEFined: user defined

Global Repeated Capabilities: repcap.Carrier

set_band(*operation_band: RsCmwWcdmaSig.enums.OperationBand*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CARRier<carrier>:BAND
driver.configure.carrier.set_band(operation_band = enums.OperationBand.OB1)
```

Selects the operating band (OB) . In single-band scenarios, all carriers use the same band. If you change it for one carrier, it is also changed for the other carriers.

param operation_band OB1 | … | OB14 | OB19 | … | OB22 | OB25 | OB26 | OB32 | OBS1 | … | OBS3 | OBL1 | UDEFined OB1, …, OB14: operating band I to XIV OB19, …, OB22: operating band XIX to XXII OB25, OB26: operating band XXV, XXVI OB32: operating band XXXII (restricted to dual band scenarios) OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L UDEFined: user defined

Global Repeated Capabilities: repcap.Carrier

7.1.7 IqIn

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:IQIN:CARRier<Carrier>
```

class IqIn

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

class CarrierStruct

Structure for reading output 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_carrier() → CarrierStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:IQIN:CARRier<carrier>
value: CarrierStruct = driver.configure.iqIn.get_carrier()
```

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

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

Global Repeated Capabilities: repcap.Carrier

set_carrier(*value: RsCmwWcdmaSig.Implementations.Configure_.IqIn.IqIn.CarrierStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:IQIN:CARRier<carrier>
driver.configure.iqIn.set_carrier(value = CarrierStruct())
```

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

param value see the help for CarrierStruct structure arguments.

Global Repeated Capabilities: repcap.Carrier

7.1.8 Downlink

class Downlink

Downlink commands group definition. 58 total commands, 5 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.8.1 Carrier

class Carrier

Carrier commands group definition. 29 total commands, 5 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.8.1.1 Ocns

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:OCNS:TYPE
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:OCNS:LEVel
```

class Ocns

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

get_level() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:OCNS:LEVel
value: float = driver.configure.downlink.carrier.ocns.get_level()
```

Queries the total OCNS channel power (relative to the base level of the generator) . If no OCNS channels are present, INV is returned.

return level: Range: -99 dB to 0 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

get_type_py() → RsCmwWcdmaSig.enums.OcnsChannelType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:OCNS:TYPE
value: enums.OcnsChannelType = driver.configure.downlink.carrier.ocns.get_type_
↪py()
```

Selects the type of OCNS channels to be generated, see 'Orthogonal Channel Noise Simulator (OCNS) '. You can select the type manually or use the automatic mode.

return type_py: R99 | R5 | R6 | R7 | AUTO

Global Repeated Capabilities: repcap.Carrier

set_type_py(*type_py: RsCmwWcdmaSig.enums.OcnsChannelType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:OCNS:TYPE
driver.configure.downlink.carrier.ocns.set_type_py(type_py = enums.
↪OcnsChannelType.AUTO)
```

Selects the type of OCNS channels to be generated, see 'Orthogonal Channel Noise Simulator (OCNS) '. You can select the type manually or use the automatic mode.

param type_py R99 | R5 | R6 | R7 | AUTO

Global Repeated Capabilities: repcap.Carrier

7.1.8.1.2 Level

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:APOWer
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:EHICh
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:ERGCh
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:EAGCh
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:HSPDsch
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:PSCH
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:SSCH
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:PCPich
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:PCCPch
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:FDPCh
```

class Level

Level commands group definition. 11 total commands, 1 Sub-groups, 10 group commands

get_apower() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:APOWer
value: float = driver.configure.downlink.carrier.level.get_apower()
```

Queries the accumulated power (total power of all active channels relative to the base level of the generator) .

return power: Range: -80 dB to 10 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

get_eagch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:EAGCh
value: float or bool = driver.configure.downlink.carrier.level.get_eagch()
```

Sets the level of the E-AGCH. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

get_ehich() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:EHICh
value: float or bool = driver.configure.downlink.carrier.level.get_ehich()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel indicated by the last mnemonic. E-HICH and E-RGCH use the same power level. Setting the level for one channel sets the same level for the other channel. Disabling the E-HICH disables also the E-RGCH. Enabling the E-RGCH enables also the E-HICH.

return level: Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

get_ergch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:ERGCh
value: float or bool = driver.configure.downlink.carrier.level.get_ergch()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel indicated by the last mnemonic. E-HICH and E-RGCH use the same power level. Setting the level for one channel sets the same level for the other channel. Disabling the E-HICH disables also the E-RGCH. Enabling the E-RGCH enables also the E-HICH.

return level: Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

get_fdpch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:FDPCh
value: float or bool = driver.configure.downlink.carrier.level.get_fdpch()
```

Sets the level of F-DPCH. The settings of DPCH level and F-DPCH level are equal. F-DPCH is activated instead of DPCH while the CPC feature is active or while a secondary uplink is enabled

return level: Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

get_hs_pdsch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:HSPDsch
value: float or bool = driver.configure.downlink.carrier.level.get_hs_pdsch()
```

Sets the level of the HS-PDSCH summed over all active codes. Setting a power level also enables the channel.

return level: Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disable | enable the channel)

Global Repeated Capabilities: repcap.Carrier

get_pccpch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:PCCPch
value: float or bool = driver.configure.downlink.carrier.level.get_pccpch()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

get_pcpich() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:PCPich
value: float = driver.configure.downlink.carrier.level.get_pcpich()
```

Sets the level of the P-CPICH.

return level: Range: -80 dB to 0 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

get_psch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:PSCH
value: float or bool = driver.configure.downlink.carrier.level.get_psch()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

get_ssch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:SSCH
value: float or bool = driver.configure.downlink.carrier.level.get_ssch()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

set_eagch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:EAGCh
driver.configure.downlink.carrier.level.set_eagch(level = 1.0)
```

Sets the level of the E-AGCH. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

set_ehich(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:EHICh
driver.configure.downlink.carrier.level.set_ehich(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel indicated by the last mnemonic. E-HICH and E-RGCH use the same power level. Setting the level for one channel sets the same level for the other channel. Disabling the E-HICH disables also the E-RGCH. Enabling the E-RGCH enables also the E-HICH.

param level Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

set_ergch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:ERGCh
driver.configure.downlink.carrier.level.set_ergch(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel indicated by the last mnemonic. E-HICH and E-RGCH use the same power level. Setting the level for one channel sets the same level for the other channel. Disabling the E-HICH disables also the E-RGCH. Enabling the E-RGCH enables also the E-HICH.

param level Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

set_fdpch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:FDPCh
driver.configure.downlink.carrier.level.set_fdpch(level = 1.0)
```

Sets the level of F-DPCH. The settings of DPCH level and F-DPCH level are equal. F-DPCH is activated instead of DPCH while the CPC feature is active or while a secondary uplink is enabled

param level Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

set_hs_pdsch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:HSPDsch
driver.configure.downlink.carrier.level.set_hs_pdsch(level = 1.0)
```

Sets the level of the HS-PDSCH summed over all active codes. Setting a power level also enables the channel.

param level Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disable | enable the channel)

Global Repeated Capabilities: repcap.Carrier

set_pccpch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:PCCPch
driver.configure.downlink.carrier.level.set_pccpch(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

set_pcpich(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:PCPich
driver.configure.downlink.carrier.level.set_pcpich(level = 1.0)
```

Sets the level of the P-CPICH.

param level Range: -80 dB to 0 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

set_psch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:PSCH
driver.configure.downlink.carrier.level.set_psch(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

set_ssch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:SSCH
driver.configure.downlink.carrier.level.set_ssch(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.8.1.2.1 Hsscch<HSSCch>

RepCap Settings

```
# Range: No1 .. No4
rc = driver.configure.downlink.carrier.level.hsscch.repcap_hSSCch_get()
driver.configure.downlink.carrier.level.hsscch.repcap_hSSCch_set(repcap.HSSCch.No1)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:LEVel:HSSCch<HSSCch>
```

class Hsscch

Hsscch commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: HSSCch, default value after init: HSSCch.No1

get(*hSSCch=<HSSCch.Default: -1>*) → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:HSSCch<nr>
value: float or bool = driver.configure.downlink.carrier.level.hsscch.
↪get(hSSCch = repcap.HSSCch.Default)
```

Sets the level of an HS-SCCH channel. Setting a power level also enables the channel.

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

return level: Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the channel)

set(*level: float*, *hSSCch=<HSSCch.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:LEVel:HSSCch<nr>
driver.configure.downlink.carrier.level.hsscch.set(level = 1.0, hSSCch = repcap.
↪HSSCch.Default)
```

Sets the level of an HS-SCCH channel. Setting a power level also enables the channel.

param level Range: -80 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the channel)

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

Cloning the Group

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

7.1.8.1.3 Code

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:CODE:CONFlict
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:CODE:HSPDsch
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:CODE:EAGCh
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:CODE:ERGCh
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:CODE:EHICh
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:CODE:PCPich
```

class Code

Code commands group definition. 7 total commands, 1 Sub-groups, 6 group commands

class ConflictStruct

Structure for reading output parameters. Fields:

- Ocns: bool: OFF | ON
- Pcpich: bool: OFF | ON
- Scpich: bool: OFF | ON
- Pccpch: bool: OFF | ON
- Sccpch: bool: OFF | ON
- Pich: bool: OFF | ON
- Aich: bool: OFF | ON
- Dpch: bool: OFF | ON
- Hsscch_1: bool: OFF | ON
- Hsscch_2: bool: OFF | ON
- Hsscch_3: bool: OFF | ON
- Hsscch_4: bool: OFF | ON
- Hs_Pdsch: bool: OFF | ON
- Eagch: bool: OFF | ON
- Ehich: bool: OFF | ON
- Ergch: bool: OFF | ON
- Fdpch: bool: OFF | ON

get_conflict() → ConflictStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:CONFlict
value: ConflictStruct = driver.configure.downlink.carrier.code.get_conflict()


    INTRO_CMD_HELP: Queries the channelization code conflict status of the␣
↪physical channels:

    - OFF: channel causes no code conflict
```

(continues on next page)

(continued from previous page)

```
    - ON: code settings of this channel conflict with the code settings of␣
↪another channel

    :return: structure: for return value, see the help for ConflictStruct␣
↪structure arguments.
```

Global Repeated Capabilities: repcap.Carrier

get_eagch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:EAGCh
value: int = driver.configure.downlink.carrier.code.get_eagch()
```

Sets the channelization code number of the E-AGCH.

return channel_code: Range: 0 to 255

Global Repeated Capabilities: repcap.Carrier

get_ehich() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:EHICh
value: int = driver.configure.downlink.carrier.code.get_ehich()
```

E-HICH and E-RGCH use the same channelization code number. Any of the two commands sets the channelization code number for both channels.

return channel_code: Range: 0 to 127

Global Repeated Capabilities: repcap.Carrier

get_ergch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:ERGCh
value: int = driver.configure.downlink.carrier.code.get_ergch()
```

E-HICH and E-RGCH use the same channelization code number. Any of the two commands sets the channelization code number for both channels.

return channel_code: Range: 0 to 127

Global Repeated Capabilities: repcap.Carrier

get_hs_pdsch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:HSPDsch
value: int = driver.configure.downlink.carrier.code.get_hs_pdsch()
```

Sets the first channelization code number of the HS-PDSCH. The number of assigned codes depends on the HSDPA channel configuration. For a fixed reference channel for example, it depends on the H-Set. For a user-defined channel, the number is configured directly.

return channel_code: Range: 0 to 16 - number of assigned codes

Global Repeated Capabilities: repcap.Carrier

get_pcpich() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:PCPich
value: int = driver.configure.downlink.carrier.code.get_pcpich()
```

Queries the channelization code number of the P-CPICH.

return channel_code: The returned value is fixed. Range: 0

Global Repeated Capabilities: repcap.Carrier

set_eagch(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:EAGCh
driver.configure.downlink.carrier.code.set_eagch(channel_code = 1)
```

Sets the channelization code number of the E-AGCH.

param channel_code Range: 0 to 255

Global Repeated Capabilities: repcap.Carrier

set_ehich(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:EHICh
driver.configure.downlink.carrier.code.set_ehich(channel_code = 1)
```

E-HICH and E-RGCH use the same channelization code number. Any of the two commands sets the channelization code number for both channels.

param channel_code Range: 0 to 127

Global Repeated Capabilities: repcap.Carrier

set_ergch(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:ERGCh
driver.configure.downlink.carrier.code.set_ergch(channel_code = 1)
```

E-HICH and E-RGCH use the same channelization code number. Any of the two commands sets the channelization code number for both channels.

param channel_code Range: 0 to 127

Global Repeated Capabilities: repcap.Carrier

set_hs_pdsch(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:HSPDsch
driver.configure.downlink.carrier.code.set_hs_pdsch(channel_code = 1)
```

Sets the first channelization code number of the HS-PDSCH. The number of assigned codes depends on the HSDPA channel configuration. For a fixed reference channel for example, it depends on the H-Set. For a user-defined channel, the number is configured directly.

param channel_code Range: 0 to 16 - number of assigned codes

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.8.1.3.1 Hsscch<HSSCch>

RepCap Settings

```
# Range: No1 .. No4
rc = driver.configure.downlink.carrier.code.hsscch.repcap_hSSCch_get()
driver.configure.downlink.carrier.code.hsscch.repcap_hSSCch_set(repcap.HSSCch.No1)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:CODE:HSSCch<HSSCch>
```

class Hsscch

Hsscch commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: HSSCch, default value after init: HSSCch.No1

get(*hSSCch=<HSSCch.Default: -1>*) → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:HSSCch<nr>
value: int = driver.configure.downlink.carrier.code.hsscch.get(hSSCch = repcap.
↪HSSCch.Default)
```

Sets the channelization code number of an HS-SCCH channel.

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

return channel_code: Range: 0 to 127

set(*channel_code: int*, *hSSCch=<HSSCch.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:CODE:HSSCch<nr>
driver.configure.downlink.carrier.code.hsscch.set(channel_code = 1, hSSCch =␣
↪repcap.HSSCch.Default)
```

Sets the channelization code number of an HS-SCCH channel.

param channel_code Range: 0 to 127

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

Cloning the Group

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

7.1.8.1.4 Enhanced

class Enhanced

Enhanced commands group definition. 7 total commands, 4 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.8.1.4.1 Dpch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:DPCH:FSFormat
```

class Dpch

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

get_fs_format() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:DPCH:FSFormat
value: int = driver.configure.downlink.carrier.enhanced.dpch.get_fs_format()
```

Sets F-DPCH slot format according to 3GPP TS 25.211, table 16C.

return slot_format: Range: 0 to 9

Global Repeated Capabilities: repcap.Carrier

set_fs_format(*slot_format: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:DPCH:FSFormat
driver.configure.downlink.carrier.enhanced.dpch.set_fs_format(slot_format = 1)
```

Sets F-DPCH slot format according to 3GPP TS 25.211, table 16C.

param slot_format Range: 0 to 9

Global Repeated Capabilities: repcap.Carrier

7.1.8.1.4.2 Pcpich

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:PCPich:SLEVel
```

class Pcpich

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

get_slevel() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:PCPich:SLEVel
value: float = driver.configure.downlink.carrier.enhanced.pcpich.get_slevel()
```

Defines the P-CPICH power level to be reported to the UE.

return signalled_level: Range: -10 dBm to 50 dBm, Unit: dBm

Global Repeated Capabilities: repcap.Carrier

set_slevel(*signalled_level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:PCPich:SLEVel
driver.configure.downlink.carrier.enhanced.pcpich.set_slevel(signalled_level =␣
↪1.0)
```

Defines the P-CPICH power level to be reported to the UE.

param signalled_level Range: -10 dBm to 50 dBm, Unit: dBm

Global Repeated Capabilities: repcap.Carrier

7.1.8.1.4.3 HsPdsch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:HSPDsch:USFRames
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:HSPDsch:POFFset
```

class HsPdsch

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

class PoffsetStruct

Structure for reading output parameters. Fields:

- Control: enums.AutoManualMode: AUTO | MANual AUTO: The correct value is calculated automatically. MANual: The value is set manually via the parameter PwrOffsetManual.
- Pwr_Offset_Manual: float: Range: -6 dB to 13 dB, Unit: dB

get_poffset() → PoffsetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSPDsch:POFFset
value: PoffsetStruct = driver.configure.downlink.carrier.enhanced.hsPdsch.get_
↪poffset()
```

Selects whether the measurement power offset is set manually or calculated automatically. Optionally a second parameter can be sent to modify the manual power offset value. It is not relevant for automatic calculation.

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

Global Repeated Capabilities: repcap.Carrier

get_us_frames() → RsCmwWcdmaSig.enums.UnscheduledTransType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSPDsch:USFRames
value: enums.UnscheduledTransType = driver.configure.downlink.carrier.enhanced.
↪hsPdsch.get_us_frames()
```

Defines the transmission in unscheduled HS-DSCH subframes.

return type_py: DUMMy | DTX DUMMy: maintain the HS-DSCH power by sending dummy data DTX: switch off the output power

Global Repeated Capabilities: repcap.Carrier

set_poffset(*value: RsCmwWcdmaSig.Implementations.Configure_.Downlink_.Carrier_.Enhanced_.HsPdsch.HsPdsch.PoffsetStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSPDsch:POFFset
driver.configure.downlink.carrier.enhanced.hsPdsch.set_poffset(value =␣
↪PoffsetStruct())
```

Selects whether the measurement power offset is set manually or calculated automatically. Optionally a second parameter can be sent to modify the manual power offset value. It is not relevant for automatic calculation.

param value see the help for PoffsetStruct structure arguments.

Global Repeated Capabilities: repcap.Carrier

set_us_frames(*type_py: RsCmwWcdmaSig.enums.UnscheduledTransType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSPDsch:USFRames
driver.configure.downlink.carrier.enhanced.hsPdsch.set_us_frames(type_py =␣
↪enums.UnscheduledTransType.DTX)
```

Defines the transmission in unscheduled HS-DSCH subframes.

param type_py DUMMy | DTX DUMMy: maintain the HS-DSCH power by sending dummy data DTX: switch off the output power

Global Repeated Capabilities: repcap.Carrier

7.1.8.1.4.4 Hsscch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:HSSCch:USFRames
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:HSSCch:NUMBer
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:HSSCch:SELection
```

class Hsscch

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

get_number() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSSCch:NUMBer
value: int = driver.configure.downlink.carrier.enhanced.hsscch.get_number()
```

Configures the number of HS-SCCHs contained in the HS-SCCH set. <Number> = n means that the set contains the HS-SCCHs number 1 to n.

return number: Range: 1 to 4

Global Repeated Capabilities: repcap.Carrier

get_selection() → RsCmwWcdmaSig.enums.HsScchType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSSCch:SELection
value: enums.HsScchType = driver.configure.downlink.carrier.enhanced.hsscch.get_
↪selection()
```

Selects the HS-SCCH that carries the UE ID in scheduled subframes. The number <n> used below is set via method RsCmwWcdmaSig.Configure.Downlink.Carrier.Enhanced.Hsscch.number.

return type_py: CH1 | CH2 | CH3 | CH4 | RANDom | AUTomatic CH1 to CH4: The UE ID is transferred on the selected HS-SCCH. RANDom: The HS-SCCH for each transmission is selected at random among the channels 1 to n. AUTomatic: For a R5 connection, the UE ID is transferred on the HS-SCCH sequence 1, 2,..., n, 1, 2, and so on. For a R7/R8 connection, the UE ID is transferred on the appropriate HS-SCCH automatically selected depending on the used modulation scheme.

Global Repeated Capabilities: repcap.Carrier

get_us_frames() → RsCmwWcdmaSig.enums.UnscheduledTransType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSSCch:USFRames
value: enums.UnscheduledTransType = driver.configure.downlink.carrier.enhanced.
↪hsscch.get_us_frames()
```

Defines the transmission in unscheduled HS-SCCH subframes.

return type_py: DUMMy | DTX DUMMy: maintain HS-SCCH power and transfer dummy UE ID, see method RsCmwWcdmaSig.Configure.Downlink.Carrier.Hsscch.IdDummy.set DTX: switch off output power in unscheduled subframes

Global Repeated Capabilities: repcap.Carrier

set_number(*number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSSCch:NUMBer
driver.configure.downlink.carrier.enhanced.hsscch.set_number(number = 1)
```

Configures the number of HS-SCCHs contained in the HS-SCCH set. <Number> = n means that the set contains the HS-SCCHs number 1 to n.

param number Range: 1 to 4

Global Repeated Capabilities: repcap.Carrier

set_selection(*type_py: RsCmwWcdmaSig.enums.HsScchType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSSCch:SELection
driver.configure.downlink.carrier.enhanced.hsscch.set_selection(type_py = enums.
↪HsScchType.AUTomatic)
```

Selects the HS-SCCH that carries the UE ID in scheduled subframes. The number <n> used below is set via method RsCmwWcdmaSig.Configure.Downlink.Carrier.Enhanced.Hsscch.number.

param type_py CH1 | CH2 | CH3 | CH4 | RANDom | AUTomatic CH1 to CH4: The UE ID is transferred on the selected HS-SCCH. RANDom: The HS-SCCH for each transmission is selected at random among the channels 1 to n. AUTomatic: For a R5 connection, the UE ID is transferred on the HS-SCCH sequence 1, 2,..., n, 1, 2, and so on. For a R7/R8 connection, the UE ID is transferred on the appropriate HS-SCCH automatically selected depending on the used modulation scheme.

Global Repeated Capabilities: repcap.Carrier

set_us_frames(*type_py: RsCmwWcdmaSig.enums.UnscheduledTransType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:HSSCch:USFRames
driver.configure.downlink.carrier.enhanced.hsscch.set_us_frames(type_py = enums.
↪UnscheduledTransType.DTX)
```

Defines the transmission in unscheduled HS-SCCH subframes.

param type_py DUMMy | DTX DUMMy: maintain HS-SCCH power and transfer dummy UE ID, see method RsCmwWcdmaSig.Configure.Downlink.Carrier.Hsscch.IdDummy.set DTX: switch off output power in unscheduled subframes

Global Repeated Capabilities: repcap.Carrier

7.1.8.1.5 Hsscch<HSSCch>

RepCap Settings

```
# Range: No1 .. No4
rc = driver.configure.downlink.carrier.hsscch.repcap_hSSCch_get()
driver.configure.downlink.carrier.hsscch.repcap_hSSCch_set(repcap.HSSCch.No1)
```

class Hsscch

Hsscch commands group definition. 2 total commands, 2 Sub-groups, 0 group commands Repeated Capability: HSSCch, default value after init: HSSCch.No1

Cloning the Group

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

Subgroups

7.1.8.1.5.1 UeId

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:HSSCch<HSSCch>:UEID
```

class UeId

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

get(*hSSCch=<HSSCch.Default: -1>*) → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:HSSCch<nr>:UEID
value: float = driver.configure.downlink.carrier.hsscch.ueId.get(hSSCch =␣
↪repcap.HSSCch.Default)
```

Sets the UE identity for an HS-SCCH channel. In the current software version, only one UE ID is configured for the HS-SCCH set of one carrier. Changing the value for one channel changes also the values of the other channels.

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

return ue_id: Range: 0 (#H0) to 65535 (#HFFFF)

set(*ue_id: float*, *hSSCch=<HSSCch.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:HSSCch<nr>:UEID
driver.configure.downlink.carrier.hsscch.ueId.set(ue_id = 1.0, hSSCch = repcap.
↪HSSCch.Default)
```

Sets the UE identity for an HS-SCCH channel. In the current software version, only one UE ID is configured for the HS-SCCH set of one carrier. Changing the value for one channel changes also the values of the other channels.

param ue_id Range: 0 (#H0) to 65535 (#HFFFF)

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

7.1.8.1.5.2 IdDummy

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:HSSCch<HSSCch>:IDDummy
```

class IdDummy

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

get(*hSSCch=<HSSCch.Default: -1>*) → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:HSSCch<nr>
↪:IDDummy
value: float = driver.configure.downlink.carrier.hsscch.idDummy.get(hSSCch =␣
↪repcap.HSSCch.Default)
```

Sets the dummy UE identity to be sent in subframes which are not allocated to the UE. Individual values can be set per HS-SCCH.

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

return dummy_ueid: Range: 0 (#H0) to 65535 (#HFFFF)

set(*dummy_ueid: float*, *hSSCch=<HSSCch.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>:HSSCch<nr>
↪:IDDummy
driver.configure.downlink.carrier.hsscch.idDummy.set(dummy_ueid = 1.0, hSSCch =␣
↪repcap.HSSCch.Default)
```

Sets the dummy UE identity to be sent in subframes which are not allocated to the UE. Individual values can be set per HS-SCCH.

param dummy_ueid Range: 0 (#H0) to 65535 (#HFFFF)

Global Repeated Capabilities: repcap.Carrier

param hSSCch optional repeated capability selector. Default value: No1 (settable in the interface 'Hsscch')

7.1.8.2 Level

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:LEVel:SCPich
CONFigure:WCDMa:SIGNaling<Instance>:DL:LEVel:SCCPch
CONFigure:WCDMa:SIGNaling<Instance>:DL:LEVel:PICH
CONFigure:WCDMa:SIGNaling<Instance>:DL:LEVel:AICH
CONFigure:WCDMa:SIGNaling<Instance>:DL:LEVel:DPCH
```

class Level

Level commands group definition. 6 total commands, 1 Sub-groups, 5 group commands

get_aich() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:AICH
value: float or bool = driver.configure.downlink.level.get_aich()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

get_dpch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:DPCH
value: float or bool = driver.configure.downlink.level.get_dpch()
```

Set the level of DPCH. The settings of DPCH level and F-DPCH level are equal.

return level: Range: -80 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

get_pich() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:PICH
value: float or bool = driver.configure.downlink.level.get_pich()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

get_sccpch() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:SCCPch
value: float or bool = driver.configure.downlink.level.get_sccpch()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

get_scpich() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:SCPich
value: float or bool = driver.configure.downlink.level.get_scpich()
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

return level: Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

set_aich(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:AICH
driver.configure.downlink.level.set_aich(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

set_dpch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:DPCH
driver.configure.downlink.level.set_dpch(level = 1.0)
```

Set the level of DPCH. The settings of DPCH level and F-DPCH level are equal.

param level Range: -80 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

set_pich(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:PICH
driver.configure.downlink.level.set_pich(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

set_sccpch(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:SCCPch
driver.configure.downlink.level.set_sccpch(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

set_scpich(*level: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:SCPich
driver.configure.downlink.level.set_scpich(level = 1.0)
```

Set the level of the channel indicated by the last mnemonic. Setting a power level also activates the channel.

param level Range: -80 dB to 0 dB, AICH: -50 dB to 0 dB , Unit: dB Additional parameters: OFF | ON (disables the channel | enables the channel using the previous/default level)

Cloning the Group

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

Subgroups

7.1.8.2.1 Adjust

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:LEVel:ADJust
```

class Adjust

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:ADJust
driver.configure.downlink.level.adjust.set()
```

Corrects the power levels of all enabled channels to minimize the difference between the total power level of the channels and the base level.

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:LEVel:ADJust
driver.configure.downlink.level.adjust.set_with_opc()
```

Corrects the power levels of all enabled channels to minimize the difference between the total power level of the channels and the base level.

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

7.1.8.3 Code

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:CODE:SCPich
CONFigure:WCDMa:SIGNaling<Instance>:DL:CODE:PCCPch
CONFigure:WCDMa:SIGNaling<Instance>:DL:CODE:SCCPch
CONFigure:WCDMa:SIGNaling<Instance>:DL:CODE:PICH
CONFigure:WCDMa:SIGNaling<Instance>:DL:CODE:AICH
CONFigure:WCDMa:SIGNaling<Instance>:DL:CODE:DPCH
CONFigure:WCDMa:SIGNaling<Instance>:DL:CODE:FDPCh
```

class Code

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

get_aich() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:AICH
value: int = driver.configure.downlink.code.get_aich()
```

Set the channelization code number of the channel indicated by the last mnemonic.

return channel_code: Range: See table below

get_dpch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:DPCH
value: int = driver.configure.downlink.code.get_dpch()
```

Set the channelization code number of the channel indicated by the last mnemonic.

return channel_code: Range: See table below

get_fdpch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:FDPCh
value: int = driver.configure.downlink.code.get_fdpch()
```

Set the channelization code number of the channel indicated by the last mnemonic.

return channel_code: Range: See table below

get_pccpch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:PCCPch
value: int = driver.configure.downlink.code.get_pccpch()
```

Queries the channelization code number of the P-CCPCH.

return channel_code: The returned value is fixed. Range: 1

get_pich() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:PICH
value: int = driver.configure.downlink.code.get_pich()
```

Set the channelization code number of the channel indicated by the last mnemonic.

return channel_code: Range: See table below

get_sccpch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:SCCPch
value: int = driver.configure.downlink.code.get_sccpch()
```

Set the channelization code number of the channel indicated by the last mnemonic.

return channel_code: Range: See table below

get_scpich() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:SCPich
value: int = driver.configure.downlink.code.get_scpich()
```

Set the channelization code number of the channel indicated by the last mnemonic.

return channel_code: Range: See table below

set_aich(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:AICH
driver.configure.downlink.code.set_aich(channel_code = 1)
```

Set the channelization code number of the channel indicated by the last mnemonic.

param channel_code Range: See table below

set_dpch(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:DPCH
driver.configure.downlink.code.set_dpch(channel_code = 1)
```

Set the channelization code number of the channel indicated by the last mnemonic.

param channel_code Range: See table below

set_fdpch(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:FDPCh
driver.configure.downlink.code.set_fdpch(channel_code = 1)
```

Set the channelization code number of the channel indicated by the last mnemonic.

param channel_code Range: See table below

set_pich(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:PICH
driver.configure.downlink.code.set_pich(channel_code = 1)
```

Set the channelization code number of the channel indicated by the last mnemonic.

param channel_code Range: See table below

set_sccpch(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:SCCPch
driver.configure.downlink.code.set_sccpch(channel_code = 1)
```

Set the channelization code number of the channel indicated by the last mnemonic.

param channel_code Range: See table below

set_scpich(*channel_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:CODE:SCPich
driver.configure.downlink.code.set_scpich(channel_code = 1)
```

Set the channelization code number of the channel indicated by the last mnemonic.

param channel_code Range: See table below

7.1.8.4 Enhanced

class Enhanced

Enhanced commands group definition. 12 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.8.4.1 Dpch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:RXLStrategy
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:PHASe
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:SSCode
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:TOFFset
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:RANGe
```

class Dpch

Dpch commands group definition. 8 total commands, 1 Sub-groups, 5 group commands

class RangeStruct
Structure for reading output parameters. Fields:

- Level_Min: float: Range: -80 dB to 0 dB, Unit: dB
- Level_Max: float: Range: -80 dB to 0 dB, Unit: dB

get_phase() → RsCmwWcdmaSig.enums.PhaseReference

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:PHASe
value: enums.PhaseReference = driver.configure.downlink.enhanced.dpch.get_
↪phase()
```

Sets the DPCH phase reference. For the S-CPICH phase shift, see method RsCmwWcdmaSig.Configure.Downlink.Enhanced.Scpich. phase.

return reference: PCPich | SCPich PCPich: P-CPICH set as reference SCPich: S-CPICH set as reference

get_range() → RangeStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:RANGe
value: RangeStruct = driver.configure.downlink.enhanced.dpch.get_range()
```

Specifies the allowed range for the variation of the DPDCH/F-DPCH power level relative to the base level Ior.

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

get_rxl_strategy() → RsCmwWcdmaSig.enums.PowerStrategy

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:RXLStrategy
value: enums.PowerStrategy = driver.configure.downlink.enhanced.dpch.get_rxl_
↪strategy()
```

Specifies the algorithm for generated power DPCH level in downlink for 'WCDMA Out-Of-Sync Handling Measurement'.

return strategy: AF | BF | CE AF: 'Max A off F Max' BF: 'Max B off F Max' CE: 'Max C off E Max'

get_sscode() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:SSCode
value: int or bool = driver.configure.downlink.enhanced.dpch.get_sscode()
```

Defines index k used for calculation of a secondary scrambling code number for the DPCH/F-DPCH (see also 'Scrambling Codes') . If the secondary scrambling code is deactivated, the primary scrambling code is used (see method RsCmwWcdmaSig. Configure.Cell.Carrier.scode) .

return sec_scramb_code: Range: 1 to 15 Additional parameters: OFF | ON (disables | enables the secondary scrambling code)

get_toffset() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:TOFFset
value: float = driver.configure.downlink.enhanced.dpch.get_toffset()
```

Defines the offset between the DL P-CCPCH timing and the DL DPCH/F-DPCH timing in multiples of 256 chips (1/10 slot) .

return timing_offset: Range: 0 to 149

set_phase(*reference: RsCmwWcdmaSig.enums.PhaseReference*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:PHASe
driver.configure.downlink.enhanced.dpch.set_phase(reference = enums.
↪PhaseReference.PCPich)
```

Sets the DPCH phase reference. For the S-CPICH phase shift, see method RsCmwWcdmaSig.Configure.Downlink.Enhanced.Scpich. phase.

param reference PCPich | SCPich PCPich: P-CPICH set as reference SCPich: S-CPICH set as reference

set_range(*value: RsCmwWcdmaSig.Implementations.Configure_.Downlink_.Enhanced_.Dpch.Dpch.RangeStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:RANGe
driver.configure.downlink.enhanced.dpch.set_range(value = RangeStruct())
```

Specifies the allowed range for the variation of the DPDCH/F-DPCH power level relative to the base level Ior.

param value see the help for RangeStruct structure arguments.

set_rxl_strategy(*strategy: RsCmwWcdmaSig.enums.PowerStrategy*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:RXLStrategy
driver.configure.downlink.enhanced.dpch.set_rxl_strategy(strategy = enums.
↪PowerStrategy.AF)
```

Specifies the algorithm for generated power DPCH level in downlink for 'WCDMA Out-Of-Sync Handling Measurement'.

param strategy AF | BF | CE AF: 'Max A off F Max' BF: 'Max B off F Max' CE: 'Max C off E Max'

set_sscode(*sec_scramb_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:SSCode
driver.configure.downlink.enhanced.dpch.set_sscode(sec_scramb_code = 1)
```

Defines index k used for calculation of a secondary scrambling code number for the DPCH/F-DPCH (see also 'Scrambling Codes') . If the secondary scrambling code is deactivated, the primary scrambling code is used (see method RsCmwWcdmaSig. Configure.Cell.Carrier.scode) .

param sec_scramb_code Range: 1 to 15 Additional parameters: OFF | ON (disables | enables the secondary scrambling code)

set_toffset(*timing_offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:TOFFset
driver.configure.downlink.enhanced.dpch.set_toffset(timing_offset = 1.0)
```

Defines the offset between the DL P-CCPCH timing and the DL DPCH/F-DPCH timing in multiples of 256 chips (1/10 slot) .

param timing_offset Range: 0 to 149

Cloning the Group

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

Subgroups

7.1.8.4.1.1 Lsequence

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:LSEQuence:STATe
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:LSEQuence
```

class Lsequence

Lsequence commands group definition. 3 total commands, 1 Sub-groups, 2 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Ab: float: Range: -80 dB to 0 dB
- Bd: float: Range: -80 dB to 0 dB
- De: float: Range: -80 dB to 0 dB
- Ef: float: Range: -80 dB to 0 dB

get_state() → RsCmwWcdmaSig.enums.LevelSeqState

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:LSEQuence:STATe
value: enums.LevelSeqState = driver.configure.downlink.enhanced.dpch.lsequence.
↪get_state()
```

Queries the generator status of DPCH level transitions for 'WCDMA Out-Of-Sync Handling Measurement'.

return state: IDLE | RUNNing | FAILed | SCONflict | SCHanged IDLE: test procedure has not started yet RUNNing: test procedure is in progress without errors FAILed: test procedure failed SCONflict: settings are inappropriate for the setup SCHanged: relevant settings changed after setup execution

get_value() → ValueStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:LSEQuence
value: ValueStruct = driver.configure.downlink.enhanced.dpch.lsequence.get_
↪value()
```

Specifies the level of out-of-sync power mask between the areas A to F.

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

set_value(*value: RsCmwWcdmaSig.Implementations.Configure_.Downlink_.Enhanced_.Dpch_.Lsequence.Lsequence.ValueStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:LSEQuence
driver.configure.downlink.enhanced.dpch.lsequence.set_value(value =␣
↪ValueStruct())
```

Specifies the level of out-of-sync power mask between the areas A to F.

param value see the help for ValueStruct structure arguments.

Cloning the Group

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

Subgroups

7.1.8.4.1.2 Execute

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:DPCH:LSEQuence:EXECute
```

class Execute

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:LSEQuence:EXECute
driver.configure.downlink.enhanced.dpch.lsequence.execute.set()
```

Initiates the DPCH level transitions in downlink according to out-of-sync power mask. The function is only available during a call.

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:DPCH:LSEQuence:EXECute
driver.configure.downlink.enhanced.dpch.lsequence.execute.set_with_opc()
```

Initiates the DPCH level transitions in downlink according to out-of-sync power mask. The function is only available during a call.

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

7.1.8.4.2 Aich

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:AICH:ACKNowledge
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:AICH:TTIMing
```

class Aich

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

get_acknowledge() → RsCmwWcdmaSig.enums.SlopeType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:AICH:ACKNowledge
value: enums.SlopeType = driver.configure.downlink.enhanced.aich.get_
↪acknowledge()
```

Defines how the R&S CMW acknowledges RACH preambles received from the UE.

return acknowledge: POSitive | NEGative POSitive: The R&S CMW acknowledges or negatively acknowledges the preambles appropriately. NEGative: The R&S CMW always responds with negative acknowledgements.

get_ttiming() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:AICH:TTIMing
value: float = driver.configure.downlink.enhanced.aich.get_ttiming()
```

Defines the minimum allowed time delay between two consecutive RACH preambles.

return transm_timing: Minimum time delay Range: 3 slots to 4 slots

set_acknowledge(*acknowledge: RsCmwWcdmaSig.enums.SlopeType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:AICH:ACKNowledge
driver.configure.downlink.enhanced.aich.set_acknowledge(acknowledge = enums.
↪SlopeType.NEGative)
```

Defines how the R&S CMW acknowledges RACH preambles received from the UE.

param acknowledge POSitive | NEGative POSitive: The R&S CMW acknowledges or negatively acknowledges the preambles appropriately. NEGative: The R&S CMW always responds with negative acknowledgements.

set_ttiming(*transm_timing: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:AICH:TTIMing
driver.configure.downlink.enhanced.aich.set_ttiming(transm_timing = 1.0)
```

Defines the minimum allowed time delay between two consecutive RACH preambles.

param transm_timing Minimum time delay Range: 3 slots to 4 slots

7.1.8.4.3 Scpich

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:SCPich:PHASe
CONFigure:WCDMa:SIGNaling<Instance>:DL:ENHanced:SCPich:SSCode
```

class Scpich

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

get_phase() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:SCPich:PHASe
value: int = driver.configure.downlink.enhanced.scpich.get_phase()
```

Defines the phase of the S-CPICH in degrees, relative to the P-CPICH phase.

return phase: Range: -315 deg to 0 deg, Unit: deg

get_sscode() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:SCPich:SSCode
value: int or bool = driver.configure.downlink.enhanced.scpich.get_sscode()
```

Defines index k used for calculation of a secondary scrambling code number for the S-CPICH (see also 'Scrambling Codes') . If the secondary scrambling code is deactivated, the primary scrambling code is used (see method RsCmwWcdmaSig.Configure. Cell.Carrier.scode) .

return sec_scramb_code: Range: 1 to 15 Additional parameters: OFF | ON (disables | enables the secondary scrambling code)

set_phase(*phase: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:SCPich:PHASe
driver.configure.downlink.enhanced.scpich.set_phase(phase = 1)
```

Defines the phase of the S-CPICH in degrees, relative to the P-CPICH phase.

param phase Range: -315 deg to 0 deg, Unit: deg

set_sscode(*sec_scramb_code: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:ENHanced:SCPich:SSCode
driver.configure.downlink.enhanced.scpich.set_sscode(sec_scramb_code = 1)
```

Defines index k used for calculation of a secondary scrambling code number for the S-CPICH (see also 'Scrambling Codes') . If the secondary scrambling code is deactivated, the primary scrambling code is used (see method RsCmwWcdmaSig.Configure. Cell.Carrier.scode) .

param sec_scramb_code Range: 1 to 15 Additional parameters: OFF | ON (disables | enables the secondary scrambling code)

7.1.8.5 Pcontrol

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:DL:PCONtrol:MODE
CONFigure:WCDMa:SIGNaling<Instance>:DL:PCONtrol:STEP
CONFigure:WCDMa:SIGNaling<Instance>:DL:PCONtrol:DTQuality
CONFigure:WCDMa:SIGNaling<Instance>:DL:PCONtrol:FTERate
```

class Pcontrol

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

get_dt_quality() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:DTQuality
value: float = driver.configure.downlink.pcontrol.get_dt_quality()
```

Specifies a signaled target BLER value.

return error_rate: Range: 0.01 % to 20 %, Unit: %

get_fterate() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:FTERate
value: float = driver.configure.downlink.pcontrol.get_fterate()
```

Specifies a signaled target TPC error rate value for tests using the F-DPCH.

return error_rate: Range: 1 % to 10 %, Unit: %

get_mode() → RsCmwWcdmaSig.enums.PowerControlMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:MODE
value: enums.PowerControlMode = driver.configure.downlink.pcontrol.get_mode()
```

Selects the frequency of power adjustment in downlink.

return mode: M0 | M1 | ON | OFF Mode 0, mode 1, additional ON / OFF enables or disables power control in downlink.

get_step() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:STEP
value: float = driver.configure.downlink.pcontrol.get_step()
```

Specifies the step size of downlink power control.

return stepsize: Range: 0.5 dB to 2 dB, Unit: dB

set_dt_quality(*error_rate: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:DTQuality
driver.configure.downlink.pcontrol.set_dt_quality(error_rate = 1.0)
```

Specifies a signaled target BLER value.

param error_rate Range: 0.01 % to 20 %, Unit: %

set_fterate(*error_rate: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:FTERate
driver.configure.downlink.pcontrol.set_fterate(error_rate = 1.0)
```

Specifies a signaled target TPC error rate value for tests using the F-DPCH.

param error_rate Range: 1 % to 10 %, Unit: %

set_mode(*mode: RsCmwWcdmaSig.enums.PowerControlMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:MODE
driver.configure.downlink.pcontrol.set_mode(mode = enums.PowerControlMode.M0)
```

Selects the frequency of power adjustment in downlink.

param mode M0 | M1 | ON | OFF Mode 0, mode 1, additional ON / OFF enables or disables power control in downlink.

set_step(*stepsize: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:DL:PCONtrol:STEP
driver.configure.downlink.pcontrol.set_step(stepsize = 1.0)
```

Specifies the step size of downlink power control.

param stepsize Range: 0.5 dB to 2 dB, Unit: dB

7.1.9 Uplink

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:MUEPower
```

class Uplink

Uplink commands group definition. 48 total commands, 7 Sub-groups, 1 group commands

get_mue_power() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:MUEPower
value: float = driver.configure.uplink.get_mue_power()
```

Sets the maximum allowed output power of the UE transmitter (averaged over the transmit slot) .

return max_ue_power: Range: -50 dBm to 33 dBm, Unit: dBm

set_mue_power(*max_ue_power: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:MUEPower
driver.configure.uplink.set_mue_power(max_ue_power = 1.0)
```

Sets the maximum allowed output power of the UE transmitter (averaged over the transmit slot) .

param max_ue_power Range: -50 dBm to 33 dBm, Unit: dBm

Cloning the Group

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

Subgroups

7.1.9.1 UepClass

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:UEPClass:REPorted
CONFigure:WCDMa:SIGNaling<Instance>:UL:UEPClass:MANual
```

class UepClass

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

get_manual() → RsCmwWcdmaSig.enums.UePowerClass

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:UEPClass:MANual
value: enums.UePowerClass = driver.configure.uplink.uepClass.get_manual()
```

Configures the UE power class value to be used by the R&S CMW if no reported value is available or usage of the reported value is disabled, see method RsCmwWcdmaSig.Configure.Uplink.UepClass.reported.

return ue_power_class: PC1 | PC2 | PC3 | PC3B | PC4 Power class 1, 2, 3, 3bis, 4

get_reported() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:UEPClass:REPorted
value: bool = driver.configure.uplink.uepClass.get_reported()
```

Enable or disable usage of the UE power class value reported by the UE. When disabled, the power class value must be set manually, see method RsCmwWcdmaSig.Configure.Uplink.UepClass.manual. The manually set value is also used if no reported value is available.

return use_reported: OFF | ON

set_manual(*ue_power_class: RsCmwWcdmaSig.enums.UePowerClass*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:UEPClass:MANual
driver.configure.uplink.uepClass.set_manual(ue_power_class = enums.UePowerClass.
↪PC1)
```

(continues on next page)

(continued from previous page)

Configures the UE power class value to be used by the R&S CMW if no reported value is available or usage of the reported value is disabled, see method RsCmwWcdmaSig.Configure.Uplink.UepClass.reported.

param ue_power_class PC1 | PC2 | PC3 | PC3B | PC4 Power class 1, 2, 3, 3bis, 4

set_reported(*use_reported: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:UEPClass:REPorted
driver.configure.uplink.uepClass.set_reported(use_reported = False)
```

Enable or disable usage of the UE power class value reported by the UE. When disabled, the power class value must be set manually, see method RsCmwWcdmaSig.Configure.Uplink.UepClass.manual. The manually set value is also used if no reported value is available.

param use_reported OFF | ON

7.1.9.2 Carrier

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:CARRier<Carrier>:POFFset
CONFigure:WCDMa:SIGNaling<Instance>:UL:CARRier<Carrier>:SCODe
```

class Carrier

Carrier commands group definition. 3 total commands, 1 Sub-groups, 2 group commands

get_poffset() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:CARRier<carrier>:POFFset
value: float = driver.configure.uplink.carrier.get_poffset()
```

Sets the DPCCH power offset, used by the UE to calculate the initial DPCCH power for random access. The power offset of the carrier two is defined as the power offset between the initial DPCCH power level on UL2 and the current DPCCH power level of UL1.

return power_offset: Range: -164 dB to -6 dB for carrier one; 0 dB to 7 dB for carrier two , Unit: dB

Global Repeated Capabilities: repcap.Carrier

get_scode() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:CARRier<carrier>:SCODe
value: float = driver.configure.uplink.carrier.get_scode()
```

Sets the long code number that the UE has to use to scramble the uplink WCDMA signal.

return scrambling_code: Range: #H0 to #HFFFFFF

Global Repeated Capabilities: repcap.Carrier

set_poffset(*power_offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:CARRier<carrier>:POFFset
driver.configure.uplink.carrier.set_poffset(power_offset = 1.0)
```

Sets the DPCCH power offset, used by the UE to calculate the initial DPCCH power for random access. The power offset of the carrier two is defined as the power offset between the initial DPCCH power level on UL2 and the current DPCCH power level of UL1.

param power_offset Range: -164 dB to -6 dB for carrier one; 0 dB to 7 dB for carrier two , Unit: dB

Global Repeated Capabilities: repcap.Carrier

set_scode(*scrambling_code: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:CARRier<carrier>:SCODe
driver.configure.uplink.carrier.set_scode(scrambling_code = 1.0)
```

Sets the long code number that the UE has to use to scramble the uplink WCDMA signal.

param scrambling_code Range: #H0 to #HFFFFFF

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.9.2.1 Tpc

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:CARRier<Carrier>:TPC:TPOWer
```

class Tpc

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

get_tpower() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:CARRier<carrier>:TPC:TPOWer
value: float = driver.configure.uplink.carrier.tpc.get_tpower()
```

Specifies a target power for the target power precondition and for the closed loop setup. INTRO_CMD_HELP: The allowed range depends on the active setup:

- 0 dBm to 33 dBm for setups 'Max. Power E-DCH' and 'DC HSPA In-Band Emission'
- 50 dBm to 33 dBm for other setups

For the secondary uplink carrier it the target power is calculated as follows: Target Power (secondary carrier) = Target Power - Target Power Offset

return target_power: Range: depends on active setup, see above , Unit: dBm

Global Repeated Capabilities: repcap.Carrier

set_tpower(*target_power: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:CARRier<carrier>:TPC:TPOWer
driver.configure.uplink.carrier.tpc.set_tpower(target_power = 1.0)
```

Specifies a target power for the target power precondition and for the closed loop setup. INTRO_CMD_HELP: The allowed range depends on the active setup:

- 0 dBm to 33 dBm for setups 'Max. Power E-DCH' and 'DC HSPA In-Band Emission'
- 50 dBm to 33 dBm for other setups

For the secondary uplink carrier it the target power is calculated as follows: Target Power (secondary carrier) = Target Power - Target Power Offset

param target_power Range: depends on active setup, see above , Unit: dBm

Global Repeated Capabilities: repcap.Carrier

7.1.9.3 OlpControl

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:OLPControl:INTerference
CONFigure:WCDMa:SIGNaling<Instance>:UL:OLPControl:CVALue
```

class OlpControl

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

get_cvalue() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:OLPControl:CVALue
value: float = driver.configure.uplink.olpControl.get_cvalue()
```

Sets the constant offset value for the initial preamble power.

return con_offset_value: Range: -35 dB to -10 dB, Unit: dB

get_interference() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:OLPControl:INTerference
value: float = driver.configure.uplink.olpControl.get_interference()
```

Estimated UL interference contained in system information block type 7.

return interference: Range: -110 dBm to -70 dBm, Unit: dBm

set_cvalue(*con_offset_value: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:OLPControl:CVALue
driver.configure.uplink.olpControl.set_cvalue(con_offset_value = 1.0)
```

Sets the constant offset value for the initial preamble power.

param con_offset_value Range: -35 dB to -10 dB, Unit: dB

set_interference(*interference: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:OLPControl:INTerference
driver.configure.uplink.olpControl.set_interference(interference = 1.0)
```

Estimated UL interference contained in system information block type 7.

param interference Range: -110 dBm to -70 dBm, Unit: dBm

7.1.9.4 Prach

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:DRXCycle
```

class Prach

Prach commands group definition. 9 total commands, 2 Sub-groups, 1 group commands

get_drx_cycle() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:DRXCycle
value: int = driver.configure.uplink.prach.get_drx_cycle()
```

Specifies the DRX cycle length.

return cycle_length: Cycle length in multiples of 2 frames Range: 6 to 9

set_drx_cycle(*cycle_length: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:DRXCycle
driver.configure.uplink.prach.set_drx_cycle(cycle_length = 1)
```

Specifies the DRX cycle length.

param cycle_length Cycle length in multiples of 2 frames Range: 6 to 9

Cloning the Group

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

Subgroups

7.1.9.4.1 Preamble

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:AICH
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:SSIZe
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:SUBChannels
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:MCYCles
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:MRETrans
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:SIGNature
```

class Preamble

Preamble commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

get_aich() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:AICH
value: int = driver.configure.uplink.prach.preamble.get_aich()
```

Specifies the number of preambles to be received before the instrument transmits the AICH.

return preambles: Range: 1 to 12

get_mcycles() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:MCYCles
value: int = driver.configure.uplink.prach.preamble.get_mcycles()
```

Specifies the maximum number of times the preamble cycle is repeated.

return max_cycles: Range: 1 to 32

get_mretrans() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:MRETrans
value: int = driver.configure.uplink.prach.preamble.get_mretrans()
```

Sets the maximum number of preambles to be transmitted before a single preamble cycle is terminated.

return retransmission: Range: 1 to 64

get_signature() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:SIGNature
value: float = driver.configure.uplink.prach.preamble.get_signature()
```

Specifies which of the 16 signatures defined by 3GPP TS 25.213 are available and associated with the PRACH. The information is coded in a 16-bit number. The bits from left to right indicate the availability of signature 15 to signature 0 (0=not available, 1=available) .

return signature: Range: #B0000000000000000 to #B1111111111111111

get_ssize() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:SSIZe
value: int = driver.configure.uplink.prach.preamble.get_ssize()
```

Specifies the transmit power difference between two consecutive preambles.

return stepsize: Range: 1 dB to 8 dB, Unit: dB

get_sub_channels() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:SUBChannels
value: float = driver.configure.uplink.prach.preamble.get_sub_channels()
```

Specifies which of the 12 PRACH subchannels are available. The information is coded in a 12-bit number where the bits from left to right indicate the availability of subchannel 11 to subchannel 0 (0=not available, 1=available) . The default format is decimal, but you can also enter binary numbers (#B000000000000 to #B111111111111) .

return sub_channels: Range: #B000000000000 to #B111111111111

set_aich(*preambles: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:AICH
driver.configure.uplink.prach.preamble.set_aich(preambles = 1)
```

Specifies the number of preambles to be received before the instrument transmits the AICH.

param preambles Range: 1 to 12

set_mcycles(*max_cycles: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:MCYCles
driver.configure.uplink.prach.preamble.set_mcycles(max_cycles = 1)
```

Specifies the maximum number of times the preamble cycle is repeated.

param max_cycles Range: 1 to 32

set_mretrans(*retransmission: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:MRETrans
driver.configure.uplink.prach.preamble.set_mretrans(retransmission = 1)
```

Sets the maximum number of preambles to be transmitted before a single preamble cycle is terminated.

param retransmission Range: 1 to 64

set_signature(*signature: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:SIGNature
driver.configure.uplink.prach.preamble.set_signature(signature = 1.0)
```

Specifies which of the 16 signatures defined by 3GPP TS 25.213 are available and associated with the PRACH. The information is coded in a 16-bit number. The bits from left to right indicate the availability of signature 15 to signature 0 (0=not available, 1=available) .

param signature Range: #B0000000000000000 to #B1111111111111111

set_ssize(*stepsize: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:SSIZe
driver.configure.uplink.prach.preamble.set_ssize(stepsize = 1)
```

Specifies the transmit power difference between two consecutive preambles.

param stepsize Range: 1 dB to 8 dB, Unit: dB

set_sub_channels(*sub_channels: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:PREamble:SUBChannels
driver.configure.uplink.prach.preamble.set_sub_channels(sub_channels = 1.0)
```

Specifies which of the 12 PRACH subchannels are available. The information is coded in a 12-bit number where the bits from left to right indicate the availability of subchannel 11 to subchannel 0 (0=not available, 1=available) . The default format is decimal, but you can also enter binary numbers (#B000000000000 to #B111111111111) .

param sub_channels Range: #B000000000000 to #B111111111111

7.1.9.4.2 Message

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:MESSage:POFFset
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:MESSage:LENGth
```

class Message

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

get_length() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:MESSage:LENGth
value: float = driver.configure.uplink.prach.message.get_length()
```

Specifies the length of the RACH transmission time interval (TTI) .

return msg_part_length: Range: 0.01 s to 0.02 s, Unit: s

get_poffset() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:MESSage:POFFset
value: float = driver.configure.uplink.prach.message.get_poffset()
```

Specifies the power difference between the last preamble transmitted and the RACH message part.

return power_offset: Range: -5 dB to 10 dB, Unit: dB

set_length(*msg_part_length: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:MESSage:LENGth
driver.configure.uplink.prach.message.set_length(msg_part_length = 1.0)
```

Specifies the length of the RACH transmission time interval (TTI) .

param msg_part_length Range: 0.01 s to 0.02 s, Unit: s

set_poffset(*power_offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:PRACh:MESSage:POFFset
driver.configure.uplink.prach.message.set_poffset(power_offset = 1.0)
```

Specifies the power difference between the last preamble transmitted and the RACH message part.

param power_offset Range: -5 dB to 10 dB, Unit: dB

7.1.9.5 Gfactor

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:VIDeo
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:VOICe
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSDPa
```

class Gfactor

Gfactor commands group definition. 12 total commands, 3 Sub-groups, 3 group commands

class HsdpaStruct

Structure for reading output parameters. Fields:

- Beta_C: int: Range: 1 to 15
- Beta_D: int: Range: 1 to 15
- Delta_Ack: int: Range: 0 to 8
- Delta_Nack: int: Range: 0 to 8
- Delta_Cqi: int: Range: 0 to 8

class VideoStruct

Structure for reading output parameters. Fields:

- Beta_C: int: Range: 1 to 15
- Beta_D: int: Range: 1 to 15

class VoiceStruct

Structure for reading output parameters. Fields:

- Beta_C: int: Range: 1 to 15
- Beta_D: int: Range: 1 to 15

get_hsdpa() → HsdpaStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSDPa
value: HsdpaStruct = driver.configure.uplink.gfactor.get_hsdpa()
```

Specifies the UE gain factors and power offsets for HSDPA connections.

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

get_video() → VideoStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:VIDeo
value: VideoStruct = driver.configure.uplink.gfactor.get_video()
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for video connections.

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

get_voice() → VoiceStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:VOICe
value: VoiceStruct = driver.configure.uplink.gfactor.get_voice()
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for voice connections.

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

set_hsdpa(*value: RsCmwWcdmaSig.Implementations.Configure_.Uplink_.Gfactor.Gfactor.HsdpaStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSDPa
driver.configure.uplink.gfactor.set_hsdpa(value = HsdpaStruct())
```

Specifies the UE gain factors and power offsets for HSDPA connections.

param value see the help for HsdpaStruct structure arguments.

set_video(*value: RsCmwWcdmaSig.Implementations.Configure_.Uplink_.Gfactor.Gfactor.VideoStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:VIDeo
driver.configure.uplink.gfactor.set_video(value = VideoStruct())
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for video connections.

param value see the help for VideoStruct structure arguments.

set_voice(*value: RsCmwWcdmaSig.Implementations.Configure_.Uplink_.Gfactor.Gfactor.VoiceStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:VOICe
driver.configure.uplink.gfactor.set_voice(value = VoiceStruct())
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for voice connections.

param value see the help for VoiceStruct structure arguments.

Cloning the Group

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

Subgroups

7.1.9.5.1 Pdata<PacketData>

RepCap Settings

```
# Range: Pd8 .. Pd384
rc = driver.configure.uplink.gfactor.pdata.repcap_packetData_get()
driver.configure.uplink.gfactor.pdata.repcap_packetData_set(repcap.PacketData.Pd8)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:PDATa<PacketData>
```

class Pdata

Pdata commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: PacketData, default value after init: PacketData.Pd8

class PdataStruct

Structure for setting input parameters. Fields:

- Beta_C: int: Range: 1 to 15
- Beta_D: int: Range: 1 to 15

get(*packetData=<PacketData.Default: -1>*) → PdataStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:PDATa<nr>
value: PdataStruct = driver.configure.uplink.gfactor.pdata.get(packetData =␣
↪repcap.PacketData.Default)
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for packet data connections.

param packetData optional repeated capability selector. Default value: Pd8 (settable in the interface 'Pdata')

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

set(*structure: RsCmwWcdmaSig.Implementations.Configure_.Uplink_.Gfactor_.Pdata.Pdata.PdataStruct*, *packetData=<PacketData.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:PDATa<nr>
driver.configure.uplink.gfactor.pdata.set(value = [PROPERTY_STRUCT_NAME](),␣
↪packetData = repcap.PacketData.Default)
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for packet data connections.

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

param packetData optional repeated capability selector. Default value: Pd8 (settable in the interface 'Pdata')

Cloning the Group

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

7.1.9.5.2 Rmc<RefMeasChannel>

RepCap Settings

```
# Range: Ch1 .. Ch5
rc = driver.configure.uplink.gfactor.rmc.repcap_refMeasChannel_get()
driver.configure.uplink.gfactor.rmc.repcap_refMeasChannel_set(repcap.RefMeasChannel.Ch1)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:RMC<RefMeasChannel>
```

class Rmc

Rmc commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: RefMeasChannel, default value after init: RefMeasChannel.Ch1

class RmcStruct

Structure for setting input parameters. Fields:

- Beta_C: int: Range: 1 to 15
- Beta_D: int: Range: 1 to 15

get(*refMeasChannel=<RefMeasChannel.Default: -1>*) → RmcStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:RMC<nr>
value: RmcStruct = driver.configure.uplink.gfactor.rmc.get(refMeasChannel =␣
↪repcap.RefMeasChannel.Default)
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for RMC connections with the selected data rate.

param refMeasChannel optional repeated capability selector. Default value: Ch1 (settable in the interface 'Rmc')

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

set(*structure: RsCmwWcdmaSig.Implementations.Configure_.Uplink_.Gfactor_.Rmc.Rmc.RmcStruct*, *refMeasChannel=<RefMeasChannel.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:RMC<nr>
driver.configure.uplink.gfactor.rmc.set(value = [PROPERTY_STRUCT_NAME](),␣
↪refMeasChannel = repcap.RefMeasChannel.Default)
```

Specifies the UE gain factors c (DPCCH) and d (DPDCH) for RMC connections with the selected data rate.

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

param refMeasChannel optional repeated capability selector. Default value: Ch1 (settable in the interface 'Rmc')

Cloning the Group

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

7.1.9.5.3 Hsupa

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSUPa:EDPCch
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSUPa:DTTP
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSUPa:EDPFormula
```

class Hsupa

Hsupa commands group definition. 7 total commands, 1 Sub-groups, 3 group commands

get_dttp() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:DTTP
value: int = driver.configure.uplink.gfactor.hsupa.get_dttp()
```

Sets the offset for traffic to total pilot power. The E-DPCCH power is highest for T2TP value of 0 and lowest for value 6.

return delta_t_2_tp: Range: 0 to 6

get_edp_formula() → RsCmwWcdmaSig.enums.UeAlgorithm

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:EDPFormula
value: enums.UeAlgorithm = driver.configure.uplink.gfactor.hsupa.get_edp_
↪formula()
```

Specifies the UE algorithm for the calculation of E-DPDCH power based on the signaled reference E-TFCIs.

return formula: EXTRapolation | INTerpolation

get_edpcch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:EDPCch
value: int = driver.configure.uplink.gfactor.hsupa.get_edpcch()
```

Specifies the signaled value E-DPCCH for HSUPA.

return delta: Range: 0 to 8

set_dttp(*delta_t_2_tp: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:DTTP
driver.configure.uplink.gfactor.hsupa.set_dttp(delta_t_2_tp = 1)
```

Sets the offset for traffic to total pilot power. The E-DPCCH power is highest for T2TP value of 0 and lowest for value 6.

param delta_t_2_tp Range: 0 to 6

set_edp_formula(*formula: RsCmwWcdmaSig.enums.UeAlgorithm*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:EDPFormula
driver.configure.uplink.gfactor.hsupa.set_edp_formula(formula = enums.
↪UeAlgorithm.EXTRapolation)
```

Specifies the UE algorithm for the calculation of E-DPDCH power based on the signaled reference E-TFCIs.

param formula EXTRapolation | INTerpolation

set_edpcch(*delta: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:EDPCch
driver.configure.uplink.gfactor.hsupa.set_edpcch(delta = 1)
```

Specifies the signaled value E-DPCCH for HSUPA.

param delta Range: 0 to 8

Cloning the Group

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

Subgroups

7.1.9.5.3.1 Etfci

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSUPa:ETFCi:POFFset
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSUPa:ETFCi:REFerence
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSUPa:ETFCi:NUMBer
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:HSUPa:ETFCi:BOOSt
```

class Etfci

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

get_boost() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:BOOSt
value: int or bool = driver.configure.uplink.gfactor.hsupa.etfci.get_boost()
```

Specifies the E-TFCI threshold beyond which boosting of E-DPCCH is enabled.

return value: Range: 0 to 127 Additional ON / OFF enables or disables the E-DPCCH power boosting.

get_number() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:NUMBer
value: int = driver.configure.uplink.gfactor.hsupa.etfci.get_number()
```

Specifies how many pairs of reference E-TFCIs and assigned power offset values are signaled to the UE.

return number: Range: 1 to 8

get_poffset() → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:POFFset
value: List[int] = driver.configure.uplink.gfactor.hsupa.etfci.get_poffset()
```

Specifies the power offset values of the first n pairs of reference E-TFCIs and power offsets, with n = 1 to 8.

return power_offset: Comma-separated list of up to 8 values (30 and 31 reserved for E-TFCI boost) Range: 0 to 31

get_reference() → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:REFerence
value: List[int] = driver.configure.uplink.gfactor.hsupa.etfci.get_reference()
```

Specifies the E-TFCI values of the first n pairs of reference E-TFCIs and power offsets, with n = 1 to 8.

return etfci: Comma-separated list of up to 8 values Range: 0 to 127

set_boost(*value: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:BOOSt
driver.configure.uplink.gfactor.hsupa.etfci.set_boost(value = 1)
```

Specifies the E-TFCI threshold beyond which boosting of E-DPCCH is enabled.

param value Range: 0 to 127 Additional ON / OFF enables or disables the E-DPCCH power boosting.

set_number(*number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:NUMBer
driver.configure.uplink.gfactor.hsupa.etfci.set_number(number = 1)
```

Specifies how many pairs of reference E-TFCIs and assigned power offset values are signaled to the UE.

param number Range: 1 to 8

set_poffset(*power_offset: List[int]*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:POFFset
driver.configure.uplink.gfactor.hsupa.etfci.set_poffset(power_offset = [1, 2,␣
↪3])
```

Specifies the power offset values of the first n pairs of reference E-TFCIs and power offsets, with n = 1 to 8.

param power_offset Comma-separated list of up to 8 values (30 and 31 reserved for E-TFCI boost) Range: 0 to 31

set_reference(*etfci: List[int]*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:GFACtor:HSUPa:ETFCi:REFerence
driver.configure.uplink.gfactor.hsupa.etfci.set_reference(etfci = [1, 2, 3])
```

Specifies the E-TFCI values of the first n pairs of reference E-TFCIs and power offsets, with n = 1 to 8.

param etfci Comma-separated list of up to 8 values Range: 0 to 127

7.1.9.6 Tpc

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:STATe
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:PATTern
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:MODE
```

class Tpc

Tpc commands group definition. 9 total commands, 5 Sub-groups, 3 group commands

get_mode() → RsCmwWcdmaSig.enums.TpcMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:MODE
value: enums.TpcMode = driver.configure.uplink.tpc.get_mode()
```

Defines the power control algorithm and the TPC step size configured at the UE.

return mode: A2S1 | A1S1 | A1S2 A2S1: algorithm 2, step size 1 dB A1S1: algorithm 1, step size 1 dB A1S2: algorithm 1, step size 2 dB

get_pattern() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:PATTern
value: str = driver.configure.uplink.tpc.get_pattern()
```

Sets the 'User Defined Pattern' to be used for 'Single Pattern' and 'Continuous Pattern'.

return pattern: String to specify the pattern. Range: up to 60 zeros and ones

get_state() → RsCmwWcdmaSig.enums.TpcState

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:STATe
value: enums.TpcState = driver.configure.uplink.tpc.get_state()
```

Queries the current TPC state.

return state: IDLE | CONTinous | ALTernating | TPLocked | TPUNlocked | MAXPower | MINPower | TRANsition | SINGle | SEARching | FAILed | MRESource | SCONflict | SCHanged IDLE: no connection established CONTinuous: transmitting continuous pattern ALTernating: transmitting alternating pattern TPLocked: closed loop target power reached TPUNlocked: reaching closed loop target power failed MAXPower: maximum power reached MINPower: minimum power reached TRANsition: transition to a state, e.g. to maximum power SINGle: transmitting a single user-defined pattern Only relevant for 'Max. Power E-DCH' setup: SEARching: setup started, max power not yet reached FAILed: test procedure failed in state 'Searching' MRESource: required resources are blocked/not available SCONflict: settings are inappropriate for the setup SCHanged: relevant settings changed after setup execution

set_mode(*mode: RsCmwWcdmaSig.enums.TpcMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:MODE
driver.configure.uplink.tpc.set_mode(mode = enums.TpcMode.A1S1)
```

Defines the power control algorithm and the TPC step size configured at the UE.

param mode A2S1 | A1S1 | A1S2 A2S1: algorithm 2, step size 1 dB A1S1: algorithm 1, step size 1 dB A1S2: algorithm 1, step size 2 dB

set_pattern(*pattern: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:PATTern
driver.configure.uplink.tpc.set_pattern(pattern = '1')
```

Sets the 'User Defined Pattern' to be used for 'Single Pattern' and 'Continuous Pattern'.

param pattern String to specify the pattern. Range: up to 60 zeros and ones

Cloning the Group

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

Subgroups

7.1.9.6.1 Set

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:SET
```

class Set

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

class GetStruct

Response structure. Fields:

- Set_Type: enums.TpcSetType: CLOop | ALTernating | ALL1 | ALL0 | SALT | SAL1 | SAL0 | CONTinuous | TSE | TSF | PHUP | PHDown | TSABc | TSEF | TSGH | MPEDch | ULCM | CTFC | DHIB CLOop: 'Closed Loop' ALTernating: 'Alternating' ALL1: 'All 1' ALL0: 'All 0' SALT: 'Single Pattern + Alternating' SAL1: 'Single Pattern + All 1' SAL0: 'Single Pattern + All 0' CONTinuous: 'Continuous Pattern' TSE: 'TPC Test Step E' TSF: 'TPC Test Step F' PHUP: 'Phase Discontinuity Up' PHDown: 'Phase Discontinuity Down' TSABc: 'TPC Test Step ABC' TSEF: 'TPC Test Step EF' TSGH: 'TPC Test Step GH' MPEDch: 'Max. Power E-DCH' ULCM: 'TPC Test Step UL CM' CTFC: 'Change of TFC' DHIB: 'DC HSPA In-Band Emission'
- Precondition: enums.Condition: NONE | ALTernating | MAXPower | MINPower | TPOWer Precondition of the active setup: none, alternating up and down, maximum, minimum or target power.
- Pconfig: str: Active setup configuration information. The content depends on the setup type: - closed loop: target power in dBm - single and continuous patterns: user-defined pattern - phase discontinuity: number of repetitions - test step EF, GH: number of 0 bits - DC HSPA in-band emission: pattern selection for the carrier one and two and number of selected bits - others: presentation of the fixed pattern
- Trigger: enums.TriggerMode: ONCE | PERiodic Type of generated trigger signal. See 'Generating TPC Trigger Signals'

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:SET
value: GetStruct = driver.configure.uplink.tpc.set.get()
```

Selects the active TPC setup. A query returns also properties of the active setup.

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

set(*set_type: RsCmwWcdmaSig.enums.TpcSetType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:SET
driver.configure.uplink.tpc.set.set(set_type = enums.TpcSetType.ALL0)
```

Selects the active TPC setup. A query returns also properties of the active setup.

param set_type CLOop | ALTernating | ALL1 | ALL0 | SALT | SAL1 | SAL0 | CONTinuous | TSE | TSF | PHUP | PHDown | TSABc | TSEF | TSGH | MPEDch | ULCM | CTFC | DHIB CLOop: 'Closed Loop' ALTernating: 'Alternating' ALL1: 'All 1' ALL0: 'All 0' SALT: 'Single Pattern + Alternating' SAL1: 'Single Pattern + All 1' SAL0: 'Single Pattern + All 0' CONTinuous: 'Continuous Pattern' TSE: 'TPC Test Step E' TSF: 'TPC Test Step F' PHUP: 'Phase Discontinuity Up' PHDown: 'Phase Discontinuity Down' TSABc: 'TPC Test Step ABC' TSEF: 'TPC Test Step EF' TSGH: 'TPC Test Step GH' MPEDch: 'Max. Power E-DCH' ULCM: 'TPC Test Step UL CM' CTFC: 'Change of TFC' DHIB: 'DC HSPA In-Band Emission'

7.1.9.6.2 Tpower

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:TPOWer:OFFSet
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:TPOWer:REFerence
```

class Tpower

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

get_offset() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:TPOWer:OFFSet
value: float = driver.configure.uplink.tpc.tpower.get_offset()
```

Specifies the difference between the target power levels of carrier one and two.

return offset: Range: -10 dB to +10 dB

get_reference() → RsCmwWcdmaSig.enums.ClosedLoopPower

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:TPOWer:REFerence
value: enums.ClosedLoopPower = driver.configure.uplink.tpc.tpower.get_
↪reference()
```

Selects the type of the closed loop target power.

return reference: TOTal | DPCH TOTal: maximum total uplink power DPCH: maximum DPCH power

set_offset(*offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:TPOWer:OFFSet
driver.configure.uplink.tpc.tpower.set_offset(offset = 1.0)
```

Specifies the difference between the target power levels of carrier one and two.

param offset Range: -10 dB to +10 dB

set_reference(*reference: RsCmwWcdmaSig.enums.ClosedLoopPower*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:TPOWer:REFerence
driver.configure.uplink.tpc.tpower.set_reference(reference = enums.
↪ClosedLoopPower.DPCH)
```

Selects the type of the closed loop target power.

param reference TOTal | DPCH TOTal: maximum total uplink power DPCH: maximum DPCH power

7.1.9.6.3 Mpedch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:MPEDch:STATe
```

class Mpedch

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

class StateStruct

Structure for reading output parameters. Fields:

- Current_Etfci_1: float: Monitored 'Current E-TFCI' value of the carrier one Range: 0 to 127
- Target_Etfci_1: float: Calculated 'Target E-TFCI' value of the carrier one Range: 0 to 127
- Current_Etfci_2: float: Monitored 'Current E-TFCI' value of the carrier two Range: 0 to 127
- Target_Etfci_2: float: Calculated 'Target E-TFCI' value of the carrier two Range: 0 to 127

get_state() → StateStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:MPEDch:STATe
value: StateStruct = driver.configure.uplink.tpc.mpedch.get_state()
```

Queries the E-TFCI information for the TPC setup 'Max. Power E-DCH'.

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

7.1.9.6.4 Precondition

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:PRECondition
```

class Precondition

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:PRECondition
driver.configure.uplink.tpc.precondition.set()
```

Reach the precondition defined for the active TPC pattern setup. Corresponds to pressing the 'Precond.' button.

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:PRECondition
driver.configure.uplink.tpc.precondition.set_with_opc()
```

Reach the precondition defined for the active TPC pattern setup. Corresponds to pressing the 'Precond.' button.

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

7.1.9.6.5 Pexecute

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:PEXecute
```

class Pexecute

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:PEXecute
driver.configure.uplink.tpc.pexecute.set()
```

Execute the active TPC pattern setup. Corresponds to pressing the 'Execute' button. For pattern setups with precondition, it is recommended to press the 'Precond.' button first (method RsCmwWcdmaSig.Configure.Uplink.Tpc.Precondition.set) .

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPC:PEXecute
driver.configure.uplink.tpc.pexecute.set_with_opc()
```

Execute the active TPC pattern setup. Corresponds to pressing the 'Execute' button. For pattern setups with precondition, it is recommended to press the 'Precond.' button first (method RsCmwWcdmaSig.Configure.Uplink.Tpc.Precondition.set) .

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

7.1.9.7 Tpcset

class Tpcset

Tpcset commands group definition. 10 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.9.7.1 Pconfig

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:TSEF
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:TSGH
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:TSSegment
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:PHDown
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:PHUP
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:DHIB
```

class Pconfig

Pconfig commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

class DhibStruct

Structure for reading output parameters. Fields:

- Config: enums.PatternType: UD | DU UD: pattern for the carrier 1 starts: 11 (up) , carrier 2: 00 (down) DU: carrier 1 starts: 00 (down) , carrier 2: 11 (up)
- Repetition: int: The number of times the pattern is repeated for each carrier. Range: 1 to 20

get_dhib() → DhibStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:DHIB
value: DhibStruct = driver.configure.uplink.tpcset.pconfig.get_dhib()
```

Defines the beginning of the pattern and the number of times the pattern has to be repeated for 'DC HSPA In-Band Emission'.

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

get_phdown() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:PHDown
value: int = driver.configure.uplink.tpcset.pconfig.get_phdown()
```

Define the number of times the pattern has to be repeated for 'Phase Discontinuity Up/Down'.

return repetition: Range: 1 to 13

get_phup() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:PHUP
value: int = driver.configure.uplink.tpcset.pconfig.get_phup()
```

Define the number of times the pattern has to be repeated for 'Phase Discontinuity Up/Down'.

return repetition: Range: 1 to 13

get_ts_segment() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:TSSegment
value: bool = driver.configure.uplink.tpcset.pconfig.get_ts_segment()
```

Enables or disables segmentation for test steps E, F, G and H.

return enable: OFF | ON

get_tsef() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:TSEF
value: int = driver.configure.uplink.tpcset.pconfig.get_tsef()
```

Defines the number of 0 bits to be sent before the all 1 pattern is started for TPC setup 'TPC Test Step EF'.

return length: Range: 100 to 170

get_tsgh() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:TSGH
value: int = driver.configure.uplink.tpcset.pconfig.get_tsgh()
```

Defines the number of 0 bits to be sent before the all 1 pattern is started for TPC setup 'TPC Test Step GH'.

return length: Range: 60 to 170

set_dhib(*value: RsCmwWcdmaSig.Implementations.Configure_.Uplink_.Tpcset_.Pconfig.Pconfig.DhibStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:DHIB
driver.configure.uplink.tpcset.pconfig.set_dhib(value = DhibStruct())
```

Defines the beginning of the pattern and the number of times the pattern has to be repeated for 'DC HSPA In-Band Emission'.

param value see the help for DhibStruct structure arguments.

set_phdown(*repetition: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:PHDown
driver.configure.uplink.tpcset.pconfig.set_phdown(repetition = 1)
```

Define the number of times the pattern has to be repeated for 'Phase Discontinuity Up/Down'.

param repetition Range: 1 to 13

set_phup(*repetition: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:PHUP
driver.configure.uplink.tpcset.pconfig.set_phup(repetition = 1)
```

Define the number of times the pattern has to be repeated for 'Phase Discontinuity Up/Down'.

param repetition Range: 1 to 13

set_ts_segment(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:TSSegment
driver.configure.uplink.tpcset.pconfig.set_ts_segment(enable = False)
```

Enables or disables segmentation for test steps E, F, G and H.

param enable OFF | ON

set_tsef(*length: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:TSEF
driver.configure.uplink.tpcset.pconfig.set_tsef(length = 1)
```

Defines the number of 0 bits to be sent before the all 1 pattern is started for TPC setup 'TPC Test Step EF'.

param length Range: 100 to 170

set_tsgh(*length: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PCONfig:TSGH
driver.configure.uplink.tpcset.pconfig.set_tsgh(length = 1)
```

Defines the number of 0 bits to be sent before the all 1 pattern is started for TPC setup 'TPC Test Step GH'.

param length Range: 60 to 170

7.1.9.7.2 Precondition

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition:PHDown
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition:PHUP
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition:CONTinuous
CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition:SINGle
```

class Precondition

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

get_continuous() → RsCmwWcdmaSig.enums.Condition

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:CONTinuous
value: enums.Condition = driver.configure.uplink.tpcset.precondition.get_
↪continuous()
```

Select the precondition for 'Continuous Pattern'.

return condition: NONE | ALTernating | MAXPower | MINPower | TPOWer

get_phdown() → RsCmwWcdmaSig.enums.ConditionB

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:PHDown
value: enums.ConditionB = driver.configure.uplink.tpcset.precondition.get_
↪phdown()
```

Select the preconditions for 'Single Pattern', 'Phase Discontinuity Up' and 'Phase Discontinuity Down'.

return condition: ALTernating | MAXPower | MINPower | TPOWer

get_phup() → RsCmwWcdmaSig.enums.ConditionB

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:PHUP
value: enums.ConditionB = driver.configure.uplink.tpcset.precondition.get_phup()
```

Select the preconditions for 'Single Pattern', 'Phase Discontinuity Up' and 'Phase Discontinuity Down'.

return condition: ALTernating | MAXPower | MINPower | TPOWer

get_single() → RsCmwWcdmaSig.enums.ConditionB

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:SINGle
value: enums.ConditionB = driver.configure.uplink.tpcset.precondition.get_
↪single()
```

Select the preconditions for 'Single Pattern', 'Phase Discontinuity Up' and 'Phase Discontinuity Down'.

return condition: ALTernating | MAXPower | MINPower | TPOWer

set_continuous(*condition: RsCmwWcdmaSig.enums.Condition*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:CONTinuous
driver.configure.uplink.tpcset.precondition.set_continuous(condition = enums.
↪Condition.ALTernating)
```

Select the precondition for 'Continuous Pattern'.

param condition NONE | ALTernating | MAXPower | MINPower | TPOWer

set_phdown(*condition: RsCmwWcdmaSig.enums.ConditionB*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:PHDown
driver.configure.uplink.tpcset.precondition.set_phdown(condition = enums.
↪ConditionB.ALTernating)
```

Select the preconditions for 'Single Pattern', 'Phase Discontinuity Up' and 'Phase Discontinuity Down'.

param condition ALTernating | MAXPower | MINPower | TPOWer

set_phup(*condition: RsCmwWcdmaSig.enums.ConditionB*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:PHUP
driver.configure.uplink.tpcset.precondition.set_phup(condition = enums.
↪ConditionB.ALTernating)
```

Select the preconditions for 'Single Pattern', 'Phase Discontinuity Up' and 'Phase Discontinuity Down'.

param condition ALTernating | MAXPower | MINPower | TPOWer

set_single(*condition: RsCmwWcdmaSig.enums.ConditionB*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:UL:TPCSet:PRECondition:SINGle
driver.configure.uplink.tpcset.precondition.set_single(condition = enums.
↪ConditionB.ALTernating)
```

Select the preconditions for 'Single Pattern', 'Phase Discontinuity Up' and 'Phase Discontinuity Down'.

param condition ALTernating | MAXPower | MINPower | TPOWer

7.1.10 Connection

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:SRBData
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:UETerminate
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:CID
```

class Connection

Connection commands group definition. 43 total commands, 6 Sub-groups, 3 group commands

class SrbSataStruct

Structure for reading output parameters. Fields:

- Downlink: enums.SrbDataRate: R1K7 | R2K5 | R3K4 | R13K6 In kbit/s: 1.7, 2.5, 3.4, 13.6
- Uplink: enums.SrbDataRate: R1K7 | R2K5 | R3K4 | R13K6 In kbit/s: 1.7, 2.5, 3.4, 13.6

get_cid() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:CID
value: str = driver.configure.connection.get_cid()
```

Sets the calling party number of the R&S CMW to be displayed at the UE. Allowed characters are 0 to 9, *, #, a, b, c.

return caller_id: 1 to 20-digit ID

get_srb_sata() → SrbSataStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:SRBData
value: SrbSataStruct = driver.configure.connection.get_srb_sata()
```

Selects the SRB data rate for downlink and uplink.

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

get_ue_terminate() → RsCmwWcdmaSig.enums.TerminatingType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:UETerminate
value: enums.TerminatingType = driver.configure.connection.get_ue_terminate()
```

Selects the connection type to be used for UE terminating connections initiated by the instrument.

return type_py: VOICe | VIDeo | SRB | TEST

set_cid(*caller_id: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:CID
driver.configure.connection.set_cid(caller_id = '1')
```

Sets the calling party number of the R&S CMW to be displayed at the UE. Allowed characters are 0 to 9, *, #, a, b, c.

param caller_id 1 to 20-digit ID

set_srb_sata(*value: RsCmwWcdmaSig.Implementations.Configure_.Connection.Connection.SrbSataStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:SRBData
driver.configure.connection.set_srb_sata(value = SrbSataStruct())
```

Selects the SRB data rate for downlink and uplink.

param value see the help for SrbSataStruct structure arguments.

set_ue_terminate(*type_py: RsCmwWcdmaSig.enums.TerminatingType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:UETerminate
driver.configure.connection.set_ue_terminate(type_py = enums.TerminatingType.
↪RMC)
```

Selects the connection type to be used for UE terminating connections initiated by the instrument.

param type_py VOICe | VIDeo | SRB | TEST

Cloning the Group

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

Subgroups

7.1.10.1 Voice

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VOICe:DTX
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VOICe:SOURce
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VOICe:CODec
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VOICe:TFCI
```

class Voice

Voice commands group definition. 7 total commands, 2 Sub-groups, 4 group commands

get_codec() → RsCmwWcdmaSig.enums.VoiceCodec

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:CODec
value: enums.VoiceCodec = driver.configure.connection.voice.get_codec()
```

Selects the AMR voice codec type to be used: narrowband or wideband.

return codec: NB | WB NB: narrowband WB: wideband

get_dtx() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:DTX
value: bool = driver.configure.connection.voice.get_dtx()
```

Enables/disables speech DTX indication in downlink.

return speech_dtx_dl: OFF | ON

get_source() → RsCmwWcdmaSig.enums.VoiceSource

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:SOURce
value: enums.VoiceSource = driver.configure.connection.voice.get_source()
```

Selects the voice connection path.

return source: LOOPback | SPEech LOOPback: voice stream looped back in the R&S CMW SPEech: connection to the speech codec board

get_tfci() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:TFCI
value: bool = driver.configure.connection.voice.get_tfci()
```

Enables/disables the downlink signaling of TFCI for voice connections.

return enable: OFF | ON

set_codec(*codec: RsCmwWcdmaSig.enums.VoiceCodec*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:CODec
driver.configure.connection.voice.set_codec(codec = enums.VoiceCodec.NB)
```

Selects the AMR voice codec type to be used: narrowband or wideband.

param codec NB | WB NB: narrowband WB: wideband

set_dtx(*speech_dtx_dl: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:DTX
driver.configure.connection.voice.set_dtx(speech_dtx_dl = False)
```

Enables/disables speech DTX indication in downlink.

param speech_dtx_dl OFF | ON

set_source(*source: RsCmwWcdmaSig.enums.VoiceSource*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:SOURce
driver.configure.connection.voice.set_source(source = enums.VoiceSource.
↪LOOPback)
```

Selects the voice connection path.

param source LOOPback | SPEech LOOPback: voice stream looped back in the R&S CMW SPEech: connection to the speech codec board

set_tfci(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:TFCI
driver.configure.connection.voice.set_tfci(enable = False)
```

Enables/disables the downlink signaling of TFCI for voice connections.

param enable OFF | ON

Cloning the Group

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

Subgroups

7.1.10.1.1 Delay

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VOICe:DELay:LOOPback
```

class Delay

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

get_loopback() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:DELay:LOOPback
value: float = driver.configure.connection.voice.delay.get_loopback()
```

Defines the time that the R&S CMW waits before it loops back the received data in the loopback voice connection.

return delay: Range: 0 s to 10 s

set_loopback(*delay: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:DELay:LOOPback
driver.configure.connection.voice.delay.set_loopback(delay = 1.0)
```

Defines the time that the R&S CMW waits before it loops back the received data in the loopback voice connection.

param delay Range: 0 s to 10 s

7.1.10.1.2 Amr

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VOICe:AMR:NARRow
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VOICe:AMR:WIDE
```

class Amr

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

get_narrow() → RsCmwWcdmaSig.enums.AmrCodecModeNarrow

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:AMR:NARRow
value: enums.AmrCodecModeNarrow = driver.configure.connection.voice.amr.get_
↪narrow()
```

Selects the mode of the NB AMR codec. The basic modes support one fixed bit-rate. Mode M supports several bit-rates.

return rate: A | B | C | D | E | F | G | H | M A: 12.2 kbps B: 10.2 kbps C: 7.95 kbps D: 7.4 kbps E: 6.7 kbps F: 5.9 kbps G: 5.15 kbps H: 4.75 kbps M: A + C + F + H

get_wide() → RsCmwWcdmaSig.enums.AmrCodecModeWide

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:AMR:WIDE
value: enums.AmrCodecModeWide = driver.configure.connection.voice.amr.get_wide()
```

Selects the mode of the WB AMR codec. The basic modes support one fixed bit-rate. Mode M supports several bit-rates.

return rate: A | B | C | D | E | F | G | H | I | M | M1 | M2 A: 23.85 kbps B: 23.05 kbps C: 19.85 kbps D: 18.25 kbps E: 15.85 kbps F: 14.25 kbps G: 12.65 kbps H: 8.85 kbps I: 6.60 kbps M: G + H + I M1: E + G + H + I M2: A + G + H + I

set_narrow(*rate: RsCmwWcdmaSig.enums.AmrCodecModeNarrow*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:AMR:NARRow
driver.configure.connection.voice.amr.set_narrow(rate = enums.
↪AmrCodecModeNarrow.A)
```

Selects the mode of the NB AMR codec. The basic modes support one fixed bit-rate. Mode M supports several bit-rates.

param rate A | B | C | D | E | F | G | H | M A: 12.2 kbps B: 10.2 kbps C: 7.95 kbps D: 7.4 kbps E: 6.7 kbps F: 5.9 kbps G: 5.15 kbps H: 4.75 kbps M: A + C + F + H

set_wide(*rate: RsCmwWcdmaSig.enums.AmrCodecModeWide*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VOICe:AMR:WIDE
driver.configure.connection.voice.amr.set_wide(rate = enums.AmrCodecModeWide.A)
```

Selects the mode of the WB AMR codec. The basic modes support one fixed bit-rate. Mode M supports several bit-rates.

param rate A | B | C | D | E | F | G | H | I | M | M1 | M2 A: 23.85 kbps B: 23.05 kbps C: 19.85 kbps D: 18.25 kbps E: 15.85 kbps F: 14.25 kbps G: 12.65 kbps H: 8.85 kbps I: 6.60 kbps M: G + H + I M1: E + G + H + I M2: A + G + H + I

7.1.10.2 Tmode

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:KTLReconfig
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:TYPE
```

class Tmode

Tmode commands group definition. 15 total commands, 3 Sub-groups, 2 group commands

get_ktlre_config() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:KTLReconfig
value: bool = driver.configure.connection.tmode.get_ktlre_config()
```

Specifies whether the test loop is kept closed when the operating band or the carrier frequency is reconfigured during an established test mode connection with test loop.

return enable: OFF | ON ON: keep test loop closed OFF: open test loop, perform reconfiguration, close test loop

get_type_py() → RsCmwWcdmaSig.enums.TestModeType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:TYPE
value: enums.TestModeType = driver.configure.connection.tmode.get_type_py()
```

Selects the test mode connection type.

return type_py: RMC | HSPA | RHSPa | FACH | BTFD RMC: RMC in CS or PS domain HSPA: HSPA in PS domain RHSPa: RMC plus HSPA FACH: test using CELL_FACH state in CS domain BTFD: test using blind transport format detection

set_ktlre_config(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:KTLReconfig
driver.configure.connection.tmode.set_ktlre_config(enable = False)
```

Specifies whether the test loop is kept closed when the operating band or the carrier frequency is reconfigured during an established test mode connection with test loop.

param enable OFF | ON ON: keep test loop closed OFF: open test loop, perform reconfiguration, close test loop

set_type_py(*type_py: RsCmwWcdmaSig.enums.TestModeType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:TYPE
driver.configure.connection.tmode.set_type_py(type_py = enums.TestModeType.BTFD)
```

Selects the test mode connection type.

param type_py RMC | HSPA | RHSPa | FACH | BTFD RMC: RMC in CS or PS domain HSPA: HSPA in PS domain RHSPa: RMC plus HSPA FACH: test using CELL_FACH state in CS domain BTFD: test using blind transport format detection

Cloning the Group

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

Subgroups

7.1.10.2.1 Btfd

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:BTFD:TFORmat
```

class Btfd

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

get_tformat() → RsCmwWcdmaSig.enums.BtfdDataRate

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:BTFD:TFORmat
value: enums.BtfdDataRate = driver.configure.connection.tmode.btfd.get_tformat()
```

Selects the downlink bit rate for BTFD RMCs.

return data_rate: R1K95 | R4K75 | R5K15 | R5K9 | R6K7 | R7K4 | R7K95 | R10K2 | R12K2 Data rate in kbit/s: 1.95, 4.75, 5.15, 5.9, 6.7, 7.4, 7.95, 10.2, 12.2

set_tformat(*data_rate: RsCmwWcdmaSig.enums.BtfdDataRate*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:BTFD:TFORmat
driver.configure.connection.tmode.btfd.set_tformat(data_rate = enums.
↪BtfdDataRate.R10K2)
```

Selects the downlink bit rate for BTFD RMCs.

param data_rate R1K95 | R4K75 | R5K15 | R5K9 | R6K7 | R7K4 | R7K95 | R10K2 | R12K2 Data rate in kbit/s: 1.95, 4.75, 5.15, 5.9, 6.7, 7.4, 7.95, 10.2, 12.2

7.1.10.2.2 Rmc

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:RMC:DOMain
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:RMC:DATA
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:RMC:DLRessources
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:RMC:UCRC
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:RMC:RLCMode
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:RMC:DRATe
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:RMC:TMODe
```

class Rmc

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

class DrateStruct

Structure for reading output parameters. Fields:

- Downlink: enums.RefChannelDataRate: R12K2 | R64K | R144k | R384k R12K2: 12.2 kbps R64K: 64 kbps R144k: 144 kbps R384k: 384 kbps
- Uplink: enums.RefChannelDataRate: R12K2 | R64K | R144k | R384k | R768k R12K2: 12.2 kbps R64K: 64 kbps R144k: 144 kbps R384k: 384 kbps R768k: 768 kbps

get_data() → RsCmwWcdmaSig.enums.BitPattern

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DATA
value: enums.BitPattern = driver.configure.connection.tmode.rmc.get_data()
```

Selects the bit pattern transmitted as user information on the DTCH. Besides 'All 0', 'All 1' and 'Alternating 0101…', pseudo-random bit sequences of variable length are available.

return pattern: ALL0 | ALL1 | ALTernating | PRBS9 | PRBS11 | PRBS13 | PRBS15

get_dl_resources() → RsCmwWcdmaSig.enums.FilledBlocks

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DLRessources
value: enums.FilledBlocks = driver.configure.connection.tmode.rmc.get_dl_
↪resources()
```

Selects the percentage of DL RMC transport blocks that are filled with information bits. The percentages are rounded, indicated in one-tenth of a percent and correspond to values 1/N, indicating that out of N transport blocks, only one is fully filled with data. (N – 1) blocks are empty. Example: P0125 = 125 ‰ = 0.125 = 1/8. Each 8th block is filled.

return filled_blocks: P0031 | P0033 | P0036 | P0038 | P0042 | P0045 | P0050 | P0056 | P0062 | P0071 | P0083 | P0100 | P0125 | P0167 | P0250 | P0500 | P1000 P0031: 1/32 P0033: 1/30 P0036: 1/28 P0038: 1/26 P0042: 1/24 P0045: 1/22 P0050: 1/20 P0056: 1/18 P0062: 1/16 P0071: 1/14 P0083: 1/12 P0100: 1/10 P0125: 1/8 P0167: 1/6 P0250: 1/4 P0500: 1/2 P1000: all blocks filled

get_domain() → RsCmwWcdmaSig.enums.RmcDomain

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DOMain
value: enums.RmcDomain = driver.configure.connection.tmode.rmc.get_domain()
```

Specifies the CS or PS domain for RMC connections in test mode.

return domain: CS | PS Circuit switched, packet switched

get_drate() → DrateStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DRATe
value: DrateStruct = driver.configure.connection.tmode.rmc.get_drate()
```

Selects the information bit rate of the downlink and uplink reference channel.

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

get_rlc_mode() → RsCmwWcdmaSig.enums.RlcMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:RLCMode
value: enums.RlcMode = driver.configure.connection.tmode.rmc.get_rlc_mode()
```

Selects the RLC mode for RMC transmission with loop mode 1.

return mode: TRANsparent | ACKNowledge

get_tmode() → RsCmwWcdmaSig.enums.UtranTestMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:TMODe
value: enums.UtranTestMode = driver.configure.connection.tmode.rmc.get_tmode()
```

Selects the test mode that the UE enters after connecting to the UTRAN.

return type_py: OFF | MODE1 | MODE2 OFF: no loop MODE1: loop mode 1 MODE2: loop mode 2

get_ucrc() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:UCRC
value: bool = driver.configure.connection.tmode.rmc.get_ucrc()
```

Enables or disables the uplink cyclic redundancy check (CRC) for loop mode 2. This setting is only relevant when an RMC with symmetric DL/UL data rate is used. The setting is separate for normal signaling and reduce signaling mode. First enable or disable reduced signaling mode (see 'Cell Setup') and afterwards configure the 'Loop Mode 2 Sym. UL CRC'.

return enable: OFF | ON

set_data(*pattern: RsCmwWcdmaSig.enums.BitPattern*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DATA
driver.configure.connection.tmode.rmc.set_data(pattern = enums.BitPattern.ALL0)
```

Selects the bit pattern transmitted as user information on the DTCH. Besides 'All 0', 'All 1' and 'Alternating 0101…', pseudo-random bit sequences of variable length are available.

param pattern ALL0 | ALL1 | ALTernating | PRBS9 | PRBS11 | PRBS13 | PRBS15

set_dl_resources(*filled_blocks: RsCmwWcdmaSig.enums.FilledBlocks*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DLRessources
driver.configure.connection.tmode.rmc.set_dl_resources(filled_blocks = enums.
↪FilledBlocks.P0031)
```

Selects the percentage of DL RMC transport blocks that are filled with information bits. The percentages are rounded, indicated in one-tenth of a percent and correspond to values 1/N, indicating that out of N transport blocks, only one is fully filled with data. (N – 1) blocks are empty. Example: P0125 = 125 ‰ = 0.125 = 1/8. Each 8th block is filled.

param filled_blocks P0031 | P0033 | P0036 | P0038 | P0042 | P0045 | P0050 | P0056 | P0062 | P0071 | P0083 | P0100 | P0125 | P0167 | P0250 | P0500 | P1000 P0031: 1/32 P0033: 1/30 P0036: 1/28 P0038: 1/26 P0042: 1/24 P0045: 1/22 P0050: 1/20 P0056: 1/18 P0062: 1/16 P0071: 1/14 P0083: 1/12 P0100: 1/10 P0125: 1/8 P0167: 1/6 P0250: 1/4 P0500: 1/2 P1000: all blocks filled

set_domain(*domain: RsCmwWcdmaSig.enums.RmcDomain*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DOMain
driver.configure.connection.tmode.rmc.set_domain(domain = enums.RmcDomain.CS)
```

Specifies the CS or PS domain for RMC connections in test mode.

param domain CS | PS Circuit switched, packet switched

set_drate(*value: RsCmwWcdmaSig.Implementations.Configure_.Connection_.Tmode_.Rmc.Rmc.DrateStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:DRATe
driver.configure.connection.tmode.rmc.set_drate(value = DrateStruct())
```

Selects the information bit rate of the downlink and uplink reference channel.

param value see the help for DrateStruct structure arguments.

set_rlc_mode(*mode: RsCmwWcdmaSig.enums.RlcMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:RLCMode
driver.configure.connection.tmode.rmc.set_rlc_mode(mode = enums.RlcMode.
↪ACKNowledge)
```

Selects the RLC mode for RMC transmission with loop mode 1.

param mode TRANsparent | ACKNowledge

set_tmode(*type_py: RsCmwWcdmaSig.enums.UtranTestMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:TMODe
driver.configure.connection.tmode.rmc.set_tmode(type_py = enums.UtranTestMode.
↪MODE1)
```

Selects the test mode that the UE enters after connecting to the UTRAN.

param type_py OFF | MODE1 | MODE2 OFF: no loop MODE1: loop mode 1 MODE2: loop mode 2

set_ucrc(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:RMC:UCRC
driver.configure.connection.tmode.rmc.set_ucrc(enable = False)
```

Enables or disables the uplink cyclic redundancy check (CRC) for loop mode 2. This setting is only relevant when an RMC with symmetric DL/UL data rate is used. The setting is separate for normal signaling and reduce signaling mode. First enable or disable reduced signaling mode (see 'Cell Setup') and afterwards configure the 'Loop Mode 2 Sym. UL CRC'.

param enable OFF | ON

7.1.10.2.3 Hspa

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:HSPA:PROCedure
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:HSPA:DIRection
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:HSPA:DATA
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:HSPA:EINSertion
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:TMODe:HSPA:USDU
```

class Hspa

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

get_data() → RsCmwWcdmaSig.enums.BitPattern

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:DATA
value: enums.BitPattern = driver.configure.connection.tmode.hspa.get_data()
```

Selects the bit pattern to be transmitted as user information on the HS-DSCH. Besides 'All 0', 'All 1' and 'Alternating 0101…', pseudo-random bit sequences of variable length are available.

return pattern: ALL0 | ALL1 | ALTernating | PRBS9 | PRBS11 | PRBS13 | PRBS15

get_direction() → RsCmwWcdmaSig.enums.HspaTestModeDirection

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:DIRection
value: enums.HspaTestModeDirection = driver.configure.connection.tmode.hspa.get_
↪direction()
```

Selects the HSPA test mode direction.

return direction: HSDPa | HSPA HSDPa: HSDPA only HSPA: HSDPA + HSUPA

get_einsertion() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:EINSertion
value: float or bool = driver.configure.connection.tmode.hspa.get_einsertion()
```

Configures the rate of HS-DSCH data to be sent with an incorrect CRC value.

return error_insertion: Range: 10 % to 90 %, Unit: % Additional parameters: OFF | ON (disables the error insertion | enables the error insertion using the previous value)

get_procedure() → RsCmwWcdmaSig.enums.Procedure

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:PROCedure
value: enums.Procedure = driver.configure.connection.tmode.hspa.get_procedure()
```

Selects whether an HSPA test mode connection is set up automatically when a test mode connection is established, or can be set up manually later on.

return procedure: CSPS | CSOPs CSPS: Establish both an RMC connection in the CS domain and an HSPA test mode connection in the PS domain. CSOPs: Establish only an RMC connection in the CS domain. You can trigger an HSPA connection setup manually later on if desired.

get_usdu() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:USDU
value: int = driver.configure.connection.tmode.hspa.get_usdu()
```

Specifies the HSUPA UL RLC SDU size as an integer multiple of the HSDPA DL RLC SDU size of 2936 bits. Beside the value of 72 bits, the command accepts a continuous range of values, but sets the nearest multiple of 2936: 72 | 2936 | 5872 | 8808 | 11744 | 14680 | 17616 | 20552 | 23488 | 26424 | 29360

return size: Range: 72 bits, 2936 bits to 29360 bits , Unit: bit

set_data(*pattern: RsCmwWcdmaSig.enums.BitPattern*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:DATA
driver.configure.connection.tmode.hspa.set_data(pattern = enums.BitPattern.ALL0)
```

Selects the bit pattern to be transmitted as user information on the HS-DSCH. Besides 'All 0', 'All 1' and 'Alternating 0101…', pseudo-random bit sequences of variable length are available.

param pattern ALL0 | ALL1 | ALTernating | PRBS9 | PRBS11 | PRBS13 | PRBS15

set_direction(*direction: RsCmwWcdmaSig.enums.HspaTestModeDirection*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:DIRection
driver.configure.connection.tmode.hspa.set_direction(direction = enums.
↪HspaTestModeDirection.HSDPa)
```

Selects the HSPA test mode direction.

param direction HSDPa | HSPA HSDPa: HSDPA only HSPA: HSDPA + HSUPA

set_einsertion(*error_insertion: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:EINSertion
driver.configure.connection.tmode.hspa.set_einsertion(error_insertion = 1.0)
```

Configures the rate of HS-DSCH data to be sent with an incorrect CRC value.

param error_insertion Range: 10 % to 90 %, Unit: % Additional parameters: OFF | ON (disables the error insertion | enables the error insertion using the previous value)

set_procedure(*procedure: RsCmwWcdmaSig.enums.Procedure*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:PROCedure
driver.configure.connection.tmode.hspa.set_procedure(procedure = enums.
↪Procedure.CSOPs)
```

Selects whether an HSPA test mode connection is set up automatically when a test mode connection is established, or can be set up manually later on.

param procedure CSPS | CSOPs CSPS: Establish both an RMC connection in the CS domain and an HSPA test mode connection in the PS domain. CSOPs: Establish only an RMC connection in the CS domain. You can trigger an HSPA connection setup manually later on if desired.

set_usdu(*size: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:TMODe:HSPA:USDU
driver.configure.connection.tmode.hspa.set_usdu(size = 1)
```

Specifies the HSUPA UL RLC SDU size as an integer multiple of the HSDPA DL RLC SDU size of 2936 bits. Beside the value of 72 bits, the command accepts a continuous range of values, but sets the nearest multiple of 2936: 72 | 2936 | 5872 | 8808 | 11744 | 14680 | 17616 | 20552 | 23488 | 26424 | 29360

param size Range: 72 bits, 2936 bits to 29360 bits , Unit: bit

7.1.10.3 Packet

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:DRATe
```

class Packet

Packet commands group definition. 15 total commands, 3 Sub-groups, 1 group commands

class DrateStruct

Structure for reading output parameters. Fields:

- Downlink: enums.DataRateDownlink: R8 | R16 | R32 | R64 | R128 | R384 | HSDPa R8 to R384: 8 kbps to 384 kbps HSDPa: HSDPA connection
- Uplink: enums.DataRateUplink: R8 | R16 | R32 | R64 | R128 | R384 | HSUPa R8 to R384: 8 kbps to 384 kbps HSUPa: HSUPA connection

get_drate() → DrateStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:DRATe
value: DrateStruct = driver.configure.connection.packet.get_drate()
```

Specifies data rates for end-to-end data connections in downlink and uplink direction.

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

set_drate(*value: RsCmwWcdmaSig.Implementations.Configure_.Connection_.Packet.Packet.DrateStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:DRATe
driver.configure.connection.packet.set_drate(value = DrateStruct())
```

Specifies data rates for end-to-end data connections in downlink and uplink direction.

param value see the help for DrateStruct structure arguments.

Cloning the Group

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

Subgroups

7.1.10.3.1 Hsdpa

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:HSDPa:RWINdow
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:HSDPa:TIMer
```

class Hsdpa

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

class RwindowStruct

Structure for reading output parameters. Fields:

- Mode: enums.AutoManualMode: AUTO | MANual Automatic calculation | manual configuration of the window size
- Receiving_Window: int: Manually configured window size applicable to Mode = MANual The value is rounded to the nearest of the following values: 1 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 768 | 1024 | 1536 | 2047 | 2560 | 3072 | 3584 | 4095 Range: 1 to 4095

class TimerStruct

Structure for reading output parameters. Fields:

- Mode: enums.AutoManualMode: AUTO | MANual Automatic calculation | manual configuration of the timeout value
- T_1_Release_Timer: float: Manually configured value applicable to Mode = MANual The value is rounded to the nearest of the following values in s: 0.01 | 0.02 | 0.03 … 0.1 | 0.12 | 0.14 | 0.16 | 0.2 | 0.3 | 0.4 Range: 0.01 s to 0.4 s, Unit: s

get_rwindow() → RwindowStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:HSDPa:RWINdow
value: RwindowStruct = driver.configure.connection.packet.hsdpa.get_rwindow()
```

Specifies the size of the receiver window in the UE.

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

get_timer() → TimerStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:HSDPa:TIMer
value: TimerStruct = driver.configure.connection.packet.hsdpa.get_timer()
```

Specifies the timeout value of the reordering release timer T1.

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

set_rwindow(*value: RsCmwWcdmaSig.Implementations.Configure_.Connection_.Packet_.Hsdpa.Hsdpa.RwindowStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:HSDPa:RWINdow
driver.configure.connection.packet.hsdpa.set_rwindow(value = RwindowStruct())
```

Specifies the size of the receiver window in the UE.

param value see the help for RwindowStruct structure arguments.

set_timer(*value: RsCmwWcdmaSig.Implementations.Configure_.Connection_.Packet_.Hsdpa.Hsdpa.TimerStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:HSDPa:TIMer
driver.configure.connection.packet.hsdpa.set_timer(value = TimerStruct())
```

Specifies the timeout value of the reordering release timer T1.

param value see the help for TimerStruct structure arguments.

7.1.10.3.2 Inactivity

class Inactivity

Inactivity commands group definition. 10 total commands, 4 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.3.2.1 Dch

class Dch
Dch commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.10.3.2.2 Network

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:DCH:NETWork:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:DCH:NETWork:TIMer
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:DCH:NETWork:DSTate
```

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

get_dstate() → RsCmwWcdmaSig.enums.DestinationState

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:NETWork:DSTate
value: enums.DestinationState = driver.configure.connection.packet.inactivity.
↪dch.network.get_dstate()
```

Specifies the destination state of the UE for automatic RRC transitions. INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:DCH:NETWork:... for state CELL_DCH (network-initiated RRC transition)
- ...:DCH:UEFDormacy:... for state CELL_DCH (UE-initiated RRC transition)
- ...:FACH:... for state CELL_FACH

return dest_state: IDLE | FACH | CPCH | UPCH Idle, CELL_FACH, CELL_PCH, URA_PCH

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:NETWork:ENABle
value: bool = driver.configure.connection.packet.inactivity.dch.network.get_
↪enable()
```

Enables or disables the network-initiated automatic RRC state transitions of the UE for the originating states CELL_DCH, CELL_FACH, CELL_PCH and URA_PCH.

return enable: OFF | ON

get_timer() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:NETWork:TIMer
value: int = driver.configure.connection.packet.inactivity.dch.network.get_
↪timer()
```

Sets the timeout value for network-initiated automatic RRC state transition for originating state CELL_DCH.

return inactivity_time: Range: 1 s to 120 s, Unit: s

set_dstate(*dest_state: RsCmwWcdmaSig.enums.DestinationState*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:NETWork:DSTate
driver.configure.connection.packet.inactivity.dch.network.set_dstate(dest_state␣
↪= enums.DestinationState.CPCH)
```

Specifies the destination state of the UE for automatic RRC transitions. INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:DCH:NETWork:... for state CELL_DCH (network-initiated RRC transition)
- ...:DCH:UEFDormacy:... for state CELL_DCH (UE-initiated RRC transition)
- ...:FACH:... for state CELL_FACH

param dest_state IDLE | FACH | CPCH | UPCH Idle, CELL_FACH, CELL_PCH, URA_PCH

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:NETWork:ENABle
driver.configure.connection.packet.inactivity.dch.network.set_enable(enable =␣
↪False)
```

Enables or disables the network-initiated automatic RRC state transitions of the UE for the originating states CELL_DCH, CELL_FACH, CELL_PCH and URA_PCH.

param enable OFF | ON

set_timer(*inactivity_time: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:NETWork:TIMer
driver.configure.connection.packet.inactivity.dch.network.set_timer(inactivity_
↪time = 1)
```

Sets the timeout value for network-initiated automatic RRC state transition for originating state CELL_DCH.

param inactivity_time Range: 1 s to 120 s, Unit: s

7.1.10.3.2.3 UefDormancy

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:DCH:UEFDormancy:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:DCH:UEFDormancy:TIMer
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:DCH:UEFDormancy:DSTate
```

class UefDormancy

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

get_dstate() → RsCmwWcdmaSig.enums.DestinationState

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:UEFDormancy:DSTate
value: enums.DestinationState = driver.configure.connection.packet.inactivity.
↪dch.uefDormancy.get_dstate()
```

Specifies the destination state of the UE for automatic RRC transitions. INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:DCH:NETWork:... for state CELL_DCH (network-initiated RRC transition)
- ...:DCH:UEFDormacy:... for state CELL_DCH (UE-initiated RRC transition)
- ...:FACH:... for state CELL_FACH

return dest_state: IDLE | FACH | CPCH | UPCH Idle, CELL_FACH, CELL_PCH, URA_PCH

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:UEFDormancy:ENABle
value: bool = driver.configure.connection.packet.inactivity.dch.uefDormancy.get_
↪enable()
```

Enables or disables the UE-initiated UE fast dormancy for the UE RRC state CELL_DCH.

return enable: OFF | ON

get_timer() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:UEFDormancy:TIMer
value: int = driver.configure.connection.packet.inactivity.dch.uefDormancy.get_
↪timer()
```

Sets the T323 timeout value for UE fast dormancy.

return t_323: Range: 0 s to 120 s, Unit: s

set_dstate(*dest_state: RsCmwWcdmaSig.enums.DestinationState*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:UEFDormancy:DSTate
driver.configure.connection.packet.inactivity.dch.uefDormancy.set_dstate(dest_
↪state = enums.DestinationState.CPCH)
```

Specifies the destination state of the UE for automatic RRC transitions. INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:DCH:NETWork:... for state CELL_DCH (network-initiated RRC transition)
- ...:DCH:UEFDormacy:... for state CELL_DCH (UE-initiated RRC transition)
- ...:FACH:... for state CELL_FACH

param dest_state IDLE | FACH | CPCH | UPCH Idle, CELL_FACH, CELL_PCH, URA_PCH

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:UEFDormancy:ENABle
driver.configure.connection.packet.inactivity.dch.uefDormancy.set_enable(enable␣
↪= False)
```

Enables or disables the UE-initiated UE fast dormancy for the UE RRC state CELL_DCH.

param enable OFF | ON

set_timer(*t_323: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:DCH:UEFDormancy:TIMer
driver.configure.connection.packet.inactivity.dch.uefDormancy.set_timer(t_323 =␣
↪1)
```

Sets the T323 timeout value for UE fast dormancy.

param t_323 Range: 0 s to 120 s, Unit: s

7.1.10.3.2.4 Fach

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:FACH:TIMer
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:FACH:DSTate
```

class Fach

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

get_dstate() → RsCmwWcdmaSig.enums.DestinationState

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:FACH:DSTate
value: enums.DestinationState = driver.configure.connection.packet.inactivity.
↪fach.get_dstate()
```

Specifies the destination state of the UE for automatic RRC transitions. INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:DCH:NETWork:... for state CELL_DCH (network-initiated RRC transition)
- ...:DCH:UEFDormacy:... for state CELL_DCH (UE-initiated RRC transition)
- ...:FACH:... for state CELL_FACH

return dest_state: IDLE | FACH | CPCH | UPCH Idle, CELL_FACH, CELL_PCH, URA_PCH

get_timer() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:FACH:TIMer
value: int = driver.configure.connection.packet.inactivity.fach.get_timer()
```

Sets the timeout value for network-initiated automatic RRC state transition INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:CPCH:... for origination state CELL_PCH
- ...:FACH:... for origination state CELL_FACH
- ...:UPCH:... for origination state URA_PCH

return inactivity_time: Range: 1 s to 120 s

set_dstate(*dest_state: RsCmwWcdmaSig.enums.DestinationState*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:FACH:DSTate
driver.configure.connection.packet.inactivity.fach.set_dstate(dest_state =␣
↪enums.DestinationState.CPCH)
```

Specifies the destination state of the UE for automatic RRC transitions. INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:DCH:NETWork:... for state CELL_DCH (network-initiated RRC transition)
- ...:DCH:UEFDormacy:... for state CELL_DCH (UE-initiated RRC transition)
- ...:FACH:... for state CELL_FACH

param dest_state IDLE | FACH | CPCH | UPCH Idle, CELL_FACH, CELL_PCH, URA_PCH

set_timer(*inactivity_time: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:FACH:TIMer
driver.configure.connection.packet.inactivity.fach.set_timer(inactivity_time =␣
↪1)
```

Sets the timeout value for network-initiated automatic RRC state transition INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:CPCH:... for origination state CELL_PCH
- ...:FACH:... for origination state CELL_FACH
- ...:UPCH:... for origination state URA_PCH

param inactivity_time Range: 1 s to 120 s

7.1.10.3.2.5 Cpch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:CPCH:TIMer
```

class Cpch

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

get_timer() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:CPCH:TIMer
value: int = driver.configure.connection.packet.inactivity.cpch.get_timer()
```

Sets the timeout value for network-initiated automatic RRC state transition INTRO_CMD_HELP: The origination RRC state is indicated in the remote command name as follows:

- ...:CPCH:... for origination state CELL_PCH
- ...:FACH:... for origination state CELL_FACH
- ...:UPCH:... for origination state URA_PCH

return inactivity_time: Range: 1 s to 120 s

set_timer(*inactivity_time: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:CPCH:TIMer
driver.configure.connection.packet.inactivity.cpch.set_timer(inactivity_time =␣
↪1)
```

Sets the timeout value for network-initiated automatic RRC state transition INTRO_CMD_HELP:
The origination RRC state is indicated in the remote command name as follows:

- ...:CPCH:... for origination state CELL_PCH
- ...:FACH:... for origination state CELL_FACH
- ...:UPCH:... for origination state URA_PCH

param inactivity_time Range: 1 s to 120 s

7.1.10.3.2.6 Upch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:INACtivity:UPCH:TIMer
```

class Upch

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

get_timer() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:UPCH:TIMer
value: int = driver.configure.connection.packet.inactivity.upch.get_timer()
```

Sets the timeout value for network-initiated automatic RRC state transition INTRO_CMD_HELP:
The origination RRC state is indicated in the remote command name as follows:

- ...:CPCH:... for origination state CELL_PCH
- ...:FACH:... for origination state CELL_FACH
- ...:UPCH:... for origination state URA_PCH

return inactivity_time: Range: 1 s to 120 s

set_timer(*inactivity_time: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CONNection:PACKet:INACtivity:UPCH:TIMer
driver.configure.connection.packet.inactivity.upch.set_timer(inactivity_time =␣
↪1)
```

Sets the timeout value for network-initiated automatic RRC state transition INTRO_CMD_HELP:
The origination RRC state is indicated in the remote command name as follows:

- ...:CPCH:... for origination state CELL_PCH
- ...:FACH:... for origination state CELL_FACH
- ...:UPCH:... for origination state URA_PCH

param inactivity_time Range: 1 s to 120 s

7.1.10.3.3 Rohc

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:ROHC:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:PACKet:ROHC:PROFiles
```

class Rohc

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

class ProfilesStruct

Structure for reading output parameters. Fields:

- Profiles_0_X_001: bool: OFF | ON IP/UDP/RTP
- Profiles_0_X_002: bool: OFF | ON IP/UDP
- Profiles_0_X_004: bool: OFF | ON IP

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:ROHC:ENABle
value: bool = driver.configure.connection.packet.rohc.get_enable()
```

Enables or disables robust header compression for PS connections.

return enable: OFF | ON

get_profiles() → ProfilesStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:ROHC:PROFiles
value: ProfilesStruct = driver.configure.connection.packet.rohc.get_profiles()
```

Defines which profiles are allowed to be used by the UE in uplink.

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

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:ROHC:ENABle
driver.configure.connection.packet.rohc.set_enable(enable = False)
```

Enables or disables robust header compression for PS connections.

param enable OFF | ON

set_profiles(*value: RsCmwWcdmaSig.Implementations.Configure_.Connection_.Packet_.Rohc.Rohc.ProfilesStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:PACKet:ROHC:PROFiles
driver.configure.connection.packet.rohc.set_profiles(value = ProfilesStruct())
```

Defines which profiles are allowed to be used by the UE in uplink.

param value see the help for ProfilesStruct structure arguments.

7.1.10.4 SrbSingle

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:SRBSingle:TYPE
```

class SrbSingle

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

get_type_py() → RsCmwWcdmaSig.enums.SrbSingleType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:SRBSingle:TYPE
value: enums.SrbSingleType = driver.configure.connection.srbSingle.get_type_py()
```

Selects the radio resource control state to which the UE is commanded when an 'SRB only' connection is set up.

return type_py: CDCH | CFACh CDCH: CELL_DCH CFACh: CELL_FACH

set_type_py(*type_py: RsCmwWcdmaSig.enums.SrbSingleType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:SRBSingle:TYPE
driver.configure.connection.srbSingle.set_type_py(type_py = enums.SrbSingleType.
↪CDCH)
```

Selects the radio resource control state to which the UE is commanded when an 'SRB only' connection is set up.

param type_py CDCH | CFACh CDCH: CELL_DCH CFACh: CELL_FACH

7.1.10.5 Video

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:VIDeo:DRATe
```

class Video

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

get_drate() → RsCmwWcdmaSig.enums.VideoRate

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:VIDeo:DRATe
value: enums.VideoRate = driver.configure.connection.video.get_drate()
```

Queries the data rate for video calls.

return rate: R64K R64K: 64 kbps

7.1.10.6 Cswitched

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CONNection:CSWitched:CRELease
```

class Cswitched

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

get_crelease() → RsCmwWcdmaSig.enums.CallRelease

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:CSWitched:CRELease
value: enums.CallRelease = driver.configure.connection.cswitched.get_crelease()
```

Specifies the signaling volume during the call release.

return call_release: NORMal | LOCal NORMal: normal release LOCal: local end release without signaling

set_crelease(*call_release: RsCmwWcdmaSig.enums.CallRelease*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CONNection:CSWitched:CRELease
driver.configure.connection.cswitched.set_crelease(call_release = enums.
↪CallRelease.LOCal)
```

Specifies the signaling volume during the call release.

param call_release NORMal | LOCal NORMal: normal release LOCal: local end release without signaling

7.1.11 IhMobility

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:IHMobility:HANDover
CONFigure:WCDMa:SIGNaling<Instance>:IHMobility:MTCS
```

class IhMobility

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

get_handover() → RsCmwWcdmaSig.enums.Handover

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:IHMobility:HANDover
value: enums.Handover = driver.configure.ihMobility.get_handover()
```

Selects the connection type to be used for an inter-RAT incoming handover in the WCDMA signaling as a handover destination.

return handover: VOICe | PACKet | TM CS voice, PS data end-to-end or test mode connection.

get_mtcs() → RsCmwWcdmaSig.enums.CsFallbackConnectionType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:IHMobility:MTCS
value: enums.CsFallbackConnectionType = driver.configure.ihMobility.get_mtcs()
```

Selects the connection type to be used for an inter-RAT incoming mobile terminated CS fallback.

return type_py: VOICe | TMRMc Voice or test mode RMC connection

set_handover(*handover: RsCmwWcdmaSig.enums.Handover*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:IHMobility:HANDover
driver.configure.ihMobility.set_handover(handover = enums.Handover.PACKet)
```

Selects the connection type to be used for an inter-RAT incoming handover in the WCDMA signaling as a handover destination.

param handover VOICe | PACKet | TM CS voice, PS data end-to-end or test mode connection.

set_mtcs(*type_py: RsCmwWcdmaSig.enums.CsFallbackConnectionType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:IHMobility:MTCS
driver.configure.ihMobility.set_mtcs(type_py = enums.CsFallbackConnectionType.
↪TMRMc)
```

Selects the connection type to be used for an inter-RAT incoming mobile terminated CS fallback.

param type_py VOICe | TMRMc Voice or test mode RMC connection

7.1.12 Cell

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:MRVersion
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RSIGnaling
CONFigure:WCDMa:SIGNaling<Instance>:CELL:PSDomain
CONFigure:WCDMa:SIGNaling<Instance>:CELL:IDENtity
CONFigure:WCDMa:SIGNaling<Instance>:CELL:IDNode
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RNC
CONFigure:WCDMa:SIGNaling<Instance>:CELL:URA
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RAC
CONFigure:WCDMa:SIGNaling<Instance>:CELL:LAC
CONFigure:WCDMa:SIGNaling<Instance>:CELL:NTOPeration
CONFigure:WCDMa:SIGNaling<Instance>:CELL:MCC
CONFigure:WCDMa:SIGNaling<Instance>:CELL:BINDicator
```

class Cell

Cell commands group definition. 145 total commands, 14 Sub-groups, 12 group commands

get_bindicator() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:BINDicator
value: bool = driver.configure.cell.get_bindicator()
```

Specifies whether the band indicator has to be broadcast as part of the system information or not.

return enable: OFF | ON ON: broadcast band indicator OFF: do not broadcast band indicator

get_id_node() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:IDNode
value: float = driver.configure.cell.get_id_node()
```

Specifies the NodeB identity (16-digit binary number) .

return value: Range: #B0 to #B1111111111111111

get_identity() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:IDENtity
value: float = driver.configure.cell.get_identity()
```

Specifies the cell identity (28-digit binary number) .

return value: Range: #B0 to #B1111111111111111111111111111

get_lac() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:LAC
value: float = driver.configure.cell.get_lac()
```

Specifies the location area code for CS services.

return value: Range: #H1 to #HFFFD

get_mcc() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MCC
value: int = driver.configure.cell.get_mcc()
```

Specifies the three-digit mobile country code (MCC) . Leading zeros can be omitted.

return value: Range: 0 to 999

get_mr_version() → RsCmwWcdmaSig.enums.MaxRelVersion

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MRVersion
value: enums.MaxRelVersion = driver.configure.cell.get_mr_version()
```

Specifies the maximum release version as a cell limitation. Automatic setting respects the installed R&S CMW options and the UE capabilities.

return max_rel_version: AUTO | R99 | R5 | R6 | R7 | R8 | R9 | R10 | R11

get_nt_operation() → RsCmwWcdmaSig.enums.NtOperMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:NTOPeration
value: enums.NtOperMode = driver.configure.cell.get_nt_operation()
```

Selects the network operation mode indicating whether a Gs interface is present in the network (mode I) or not (mode II) .

return mode: M1 | M2 M1: mode I, Gs interface present M2: mode II, Gs interface not present

get_psdomain() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:PSDomain
value: bool = driver.configure.cell.get_psdomain()
```

Enables or disables the support of packet switched connections by the emulated UTRAN cell.

return enable: OFF | ON

get_rac() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RAC
value: float = driver.configure.cell.get_rac()
```

Specifies the routing area code for PS services (8-digit binary number) .

return value: Range: #B0 to #B11111111

get_rnc() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RNC
value: float = driver.configure.cell.get_rnc()
```

Specifies the radio network controller (RNC) identity (12-digit binary number) .

return value: Range: #B0 to #B111111111111

get_rsignaling() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RSIGnaling
value: bool = driver.configure.cell.get_rsignaling()
```

Enables or disables the reduced signaling mode.

return enable: OFF | ON

get_ura() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:URA
value: float = driver.configure.cell.get_ura()
```

Specifies the UTRAN registration area (URA) identity (16-digit binary number) .

return value: Range: #B0 to #B1111111111111111

set_bindicator(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:BINDicator
driver.configure.cell.set_bindicator(enable = False)
```

Specifies whether the band indicator has to be broadcast as part of the system information or not.

param enable OFF | ON ON: broadcast band indicator OFF: do not broadcast band indicator

set_id_node(*value: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:IDNode
driver.configure.cell.set_id_node(value = 1.0)
```

Specifies the NodeB identity (16-digit binary number) .

param value Range: #B0 to #B1111111111111111

set_identity(*value: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:IDENtity
driver.configure.cell.set_identity(value = 1.0)
```

Specifies the cell identity (28-digit binary number) .

param value Range: #B0 to #B11111111111111111111111111111

set_lac(*value: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:LAC
driver.configure.cell.set_lac(value = 1.0)
```

Specifies the location area code for CS services.

param value Range: #H1 to #HFFFD

set_mcc(*value: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MCC
driver.configure.cell.set_mcc(value = 1)
```

Specifies the three-digit mobile country code (MCC) . Leading zeros can be omitted.

param value Range: 0 to 999

set_mr_version(*max_rel_version: RsCmwWcdmaSig.enums.MaxRelVersion*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MRVersion
driver.configure.cell.set_mr_version(max_rel_version = enums.MaxRelVersion.AUTO)
```

Specifies the maximum release version as a cell limitation. Automatic setting respects the installed R&S CMW options and the UE capabilities.

param max_rel_version AUTO | R99 | R5 | R6 | R7 | R8 | R9 | R10 | R11

set_nt_operation(*mode: RsCmwWcdmaSig.enums.NtOperMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:NTOPeration
driver.configure.cell.set_nt_operation(mode = enums.NtOperMode.M1)
```

Selects the network operation mode indicating whether a Gs interface is present in the network (mode I) or not (mode II) .

param mode M1 | M2 M1: mode I, Gs interface present M2: mode II, Gs interface not present

set_psdomain(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:PSDomain
driver.configure.cell.set_psdomain(enable = False)
```

Enables or disables the support of packet switched connections by the emulated UTRAN cell.

param enable OFF | ON

set_rac(*value: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RAC
driver.configure.cell.set_rac(value = 1.0)
```

Specifies the routing area code for PS services (8-digit binary number) .

param value Range: #B0 to #B11111111

set_rnc(*value: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RNC
driver.configure.cell.set_rnc(value = 1.0)
```

Specifies the radio network controller (RNC) identity (12-digit binary number) .

param value Range: #B0 to #B111111111111

set_rsignaling(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RSIGnaling
driver.configure.cell.set_rsignaling(enable = False)
```

Enables or disables the reduced signaling mode.

param enable OFF | ON

set_ura(*value: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:URA
driver.configure.cell.set_ura(value = 1.0)
```

Specifies the UTRAN registration area (URA) identity (16-digit binary number) .

param value Range: #B0 to #B1111111111111111

Cloning the Group

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

Subgroups

7.1.12.1 Carrier

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:SCODe
```

class Carrier

Carrier commands group definition. 28 total commands, 3 Sub-groups, 1 group commands

get_scode() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:SCODe
value: float = driver.configure.cell.carrier.get_scode()
```

Specifies index i for calculation of the primary scrambling code number by multiplication with 16. For details, see 'Scrambling Codes'.

return code: Range: #H0 to #H1FF

Global Repeated Capabilities: repcap.Carrier

set_scode(*code: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:SCODe
driver.configure.cell.carrier.set_scode(code = 1.0)
```

Specifies index i for calculation of the primary scrambling code number by multiplication with 16. For details, see 'Scrambling Codes'.

param code Range: #H0 to #H1FF

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.12.1.1 Hsdpa

class Hsdpa
Hsdpa commands group definition. 8 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.12.1.1.1 Cqi

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:CQI:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:CQI:FIXed
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:CQI:CONFormance
```

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

get_conformance() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:CQI:CONFormance
value: int = driver.configure.cell.carrier.hsdpa.cqi.get_conformance()
```

Defines the CQI value used in the first stage of the test where the downlink transport format is fixed and the frequency distribution of the reported CQI values is calculated. To use this value, configure CONFormance via method RsCmwWcdmaSig. Configure.Cell.Hsdpa.Cqi.tindex.

return value: Range: 1 to 30

Global Repeated Capabilities: repcap.Carrier

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:CQI:ENABle
value: bool = driver.configure.cell.carrier.hsdpa.cqi.get_enable()
```

Enables or disables the multi-carrier operation for data transport via additional HS-DSCH.

return enable: OFF | ON

Global Repeated Capabilities: repcap.Carrier

get_fixed() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:CQI:FIXed
value: int = driver.configure.cell.carrier.hsdpa.cqi.get_fixed()
```

Selects the CQI table index to be used if FIXed is configured via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.tindex.

return fixed_value: Range: 1 to 30

Global Repeated Capabilities: repcap.Carrier

set_conformance(*value: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:CQI:CONFormance
driver.configure.cell.carrier.hsdpa.cqi.set_conformance(value = 1)
```

Defines the CQI value used in the first stage of the test where the downlink transport format is fixed and the frequency distribution of the reported CQI values is calculated. To use this value, configure CONFormance via method RsCmwWcdmaSig. Configure.Cell.Hsdpa.Cqi.tindex.

param value Range: 1 to 30

Global Repeated Capabilities: repcap.Carrier

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:CQI:ENABle
driver.configure.cell.carrier.hsdpa.cqi.set_enable(enable = False)
```

Enables or disables the multi-carrier operation for data transport via additional HS-DSCH.

param enable OFF | ON

Global Repeated Capabilities: repcap.Carrier

set_fixed(*fixed_value: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:CQI:FIXed
driver.configure.cell.carrier.hsdpa.cqi.set_fixed(fixed_value = 1)
```

Selects the CQI table index to be used if FIXed is configured via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.tindex.

param fixed_value Range: 1 to 30

Global Repeated Capabilities: repcap.Carrier

7.1.12.1.1.2 UserDefined

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:UDEFined:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:UDEFined:MODulation
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:UDEFined:NCODes
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:UDEFined:TTI
```

class UserDefined

UserDefined commands group definition. 5 total commands, 1 Sub-groups, 4 group commands

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:ENABle
value: bool = driver.configure.cell.carrier.hsdpa.userDefined.get_enable()
```

Enables or disables the multi-carrier operation for data transport via additional HS-DSCH.

return enable: OFF | ON

Global Repeated Capabilities: repcap.Carrier

get_modulation() → RsCmwWcdmaSig.enums.HsdpaModulation

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:MODulation
value: enums.HsdpaModulation = driver.configure.cell.carrier.hsdpa.userDefined.
↪get_modulation()
```

Selects the modulation scheme to be used.

return modulation: QPSK | Q16 | Q64 QPSK, 16-QAM, 64-QAM

Global Repeated Capabilities: repcap.Carrier

get_ncodes() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:NCODes
value: int = driver.configure.cell.carrier.hsdpa.userDefined.get_ncodes()
```

Specifies the number of HS-PDSCH channelization codes to be assigned to the UE.

return number: Range: 1 to 15

Global Repeated Capabilities: repcap.Carrier

get_tti() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:TTI
value: int = driver.configure.cell.carrier.hsdpa.userDefined.get_tti()
```

Specifies the minimum distance between two consecutive transmission time intervals in which the HS-DSCH is allocated to the UE.

return tti: Range: 1 to 3

Global Repeated Capabilities: repcap.Carrier

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:ENABle
driver.configure.cell.carrier.hsdpa.userDefined.set_enable(enable = False)
```

Enables or disables the multi-carrier operation for data transport via additional HS-DSCH.

param enable OFF | ON

Global Repeated Capabilities: repcap.Carrier

set_modulation(*modulation: RsCmwWcdmaSig.enums.HsdpaModulation*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:MODulation
driver.configure.cell.carrier.hsdpa.userDefined.set_modulation(modulation =␣
↪enums.HsdpaModulation.Q16)
```

Selects the modulation scheme to be used.

param modulation QPSK | Q16 | Q64 QPSK, 16-QAM, 64-QAM

Global Repeated Capabilities: repcap.Carrier

set_ncodes(*number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:NCODes
driver.configure.cell.carrier.hsdpa.userDefined.set_ncodes(number = 1)
```

Specifies the number of HS-PDSCH channelization codes to be assigned to the UE.

param number Range: 1 to 15

Global Repeated Capabilities: repcap.Carrier

set_tti(*tti: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:TTI
driver.configure.cell.carrier.hsdpa.userDefined.set_tti(tti = 1)
```

Specifies the minimum distance between two consecutive transmission time intervals in which the HS-DSCH is allocated to the UE.

param tti Range: 1 to 3

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.12.1.1.3 Tblock

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSDPa:UDEFined:TBLock
```

class Tblock

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

class GetStruct

Response structure. Fields:

- Index: int: Transport block size index (TFRI value) Range: 0 to 62
- Size: int: Used transport block size resulting from the settings Range: 0 bits to 28.8E+3 bits

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:TBLock
value: GetStruct = driver.configure.cell.carrier.hsdpa.userDefined.tblock.get()
```

Specifies the value of the transport format and resource indicator (TFRI) signaled to the UE. A query returns also the resulting transport block size.

Global Repeated Capabilities: repcap.Carrier

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

set(*index: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSDPa:UDEFined:TBLock
driver.configure.cell.carrier.hsdpa.userDefined.tblock.set(index = 1)
```

Specifies the value of the transport format and resource indicator (TFRI) signaled to the UE. A query returns also the resulting transport block size.

param index Transport block size index (TFRI value) Range: 0 to 62

Global Repeated Capabilities: repcap.Carrier

7.1.12.1.2 Hsupa

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:ENABle
```

class Hsupa

Hsupa commands group definition. 17 total commands, 5 Sub-groups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:HSUPa:ENABle
value: bool = driver.configure.cell.carrier.hsupa.get_enable()
```

Enables/disables additional uplink carrier in scenarios with multiple uplink carriers.

return enable: OFF | ON

Global Repeated Capabilities: repcap.Carrier

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:HSUPa:ENABle
driver.configure.cell.carrier.hsupa.set_enable(enable = False)
```

Enables/disables additional uplink carrier in scenarios with multiple uplink carriers.

param enable OFF | ON

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.12.1.2.1 Ehrch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EHRCh:FUFDummies
```

class Ehrch

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

get_fuf_dummies() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EHRCh:FUFDummies
value: bool = driver.configure.cell.carrier.hsupa.ehrch.get_fuf_dummies()
```

Enables or disables filling-up the frame with dummies. This feature is only relevant for 10 ms TTI. Here E-RGCH and E-HICH messages for the UE are transmitted in 12 slots per frame. The command defines the behavior in the remaining three slots.

return enable: OFF | ON OFF: switch off channels (DTX) ON: fill-up with dummies, continuous signal

Global Repeated Capabilities: repcap.Carrier

set_fuf_dummies(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EHRCh:FUFDummies
driver.configure.cell.carrier.hsupa.ehrch.set_fuf_dummies(enable = False)
```

Enables or disables filling-up the frame with dummies. This feature is only relevant for 10 ms TTI. Here E-RGCH and E-HICH messages for the UE are transmitted in 12 slots per frame. The command defines the behavior in the remaining three slots.

param enable OFF | ON OFF: switch off channels (DTX) ON: fill-up with dummies, continuous signal

Global Repeated Capabilities: repcap.Carrier

7.1.12.1.2.2 Eagch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EAGCh:UEID
```

class Eagch

Eagch commands group definition. 7 total commands, 1 Sub-groups, 1 group commands

class UeIdStruct

Structure for reading output parameters. Fields:

- Primary: float: Range: #H0 to #HFFFF
- Secondary: float: Range: #H0 to #HFFFF

get_ue_id() → UeIdStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:UEID
value: UeIdStruct = driver.configure.cell.carrier.hsupa.eagch.get_ue_id()
```

Specifies the primary [and secondary] E-RNTI of the UE.

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

Global Repeated Capabilities: repcap.Carrier

set_ue_id(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Carrier_.Hsupa_.Eagch.Eagch.UeIdStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:UEID
driver.configure.cell.carrier.hsupa.eagch.set_ue_id(value = UeIdStruct())
```

Specifies the primary [and secondary] E-RNTI of the UE.

param value see the help for UeIdStruct structure arguments.

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.12.1.2.3 Pattern

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EAGCh:PATTern:LENGth
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EAGCh:PATTern:INDex
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EAGCh:PATTern:SCOPe
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EAGCh:PATTern:TYPE
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EAGCh:PATTern:REPetition
```

class Pattern

Pattern commands group definition. 6 total commands, 1 Sub-groups, 5 group commands

get_index() → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:INDex
value: List[int or bool] = driver.configure.cell.carrier.hsupa.eagch.pattern.
↪get_index()
```

Specifies the absolute grant indices of the absolute grant pattern. A query returns all eight defined indices. A setting configures the first n indices (n = 1 to 8) . Only the first m indices are considered for transmission, with m specified via method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Eagch.Pattern.length.

return index: Comma-separated list of up to eight values Range: 0 to 31 Additional OFF | ON disables | enables the transmission of index value, OFF results in an unscheduled TTI

Global Repeated Capabilities: repcap.Carrier

get_length() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:LENGth
value: int = driver.configure.cell.carrier.hsupa.eagch.pattern.get_length()
```

Specifies the length of the absolute grant pattern.

return length: Range: 1 to 8 (for 10 ms TTI: 1 to 4)

Global Repeated Capabilities: repcap.Carrier

get_repetition() → RsCmwWcdmaSig.enums.RepetitionB

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:REPetition
value: enums.RepetitionB = driver.configure.cell.carrier.hsupa.eagch.pattern.
↪get_repetition()
```

Specifies whether the absolute grant pattern has to be transmitted only once, continuously or serving grant (SG) initialized. Select 'SG Initialized' only for E-RGCH measurements.

return repetition: ONCE | CONTinuous | SGINit

Global Repeated Capabilities: repcap.Carrier

get_scope() → List[bool]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:SCOPe
value: List[bool] = driver.configure.cell.carrier.hsupa.eagch.pattern.get_
↪scope()
```

Specifies the absolute grant scopes of the absolute grant pattern. A query returns all eight defined scopes. A setting configures the first n scopes (n = 1 to 8) . Only the first m scopes are considered for transmission, with m specified via method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Eagch.Pattern.length.

return scope: OFF | ON Comma-separated list of up to eight values OFF: absolute grant applies to all HARQ processes ON: absolute grant applies to one HARQ process only

Global Repeated Capabilities: repcap.Carrier

get_type_py() → List[bool]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:TYPE
value: List[bool] = driver.configure.cell.carrier.hsupa.eagch.pattern.get_type_
↪py()
```

Specifies the ID types of the absolute grant pattern. A query returns all eight defined types. A setting configures the first n types (n = 1 to 8) . Only the first m types are considered for transmission, with m specified via method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Eagch.Pattern.length.

return type_py: OFF | ON Comma-separated list of up to eight values OFF: use primary UE-ID ON: use secondary UE-ID

Global Repeated Capabilities: repcap.Carrier

set_index(*index: List[int]*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:INDex
driver.configure.cell.carrier.hsupa.eagch.pattern.set_index(index = [1, True, 2,
↪ False, 3])
```

Specifies the absolute grant indices of the absolute grant pattern. A query returns all eight defined indices. A setting configures the first n indices (n = 1 to 8) . Only the first m indices are considered for transmission, with m specified via method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Eagch.Pattern.length.

param index Comma-separated list of up to eight values Range: 0 to 31 Additional OFF | ON disables | enables the transmission of index value, OFF results in an unscheduled TTI

Global Repeated Capabilities: repcap.Carrier

set_length(*length: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:LENGth
driver.configure.cell.carrier.hsupa.eagch.pattern.set_length(length = 1)
```

Specifies the length of the absolute grant pattern.

param length Range: 1 to 8 (for 10 ms TTI: 1 to 4)

Global Repeated Capabilities: repcap.Carrier

set_repetition(*repetition: RsCmwWcdmaSig.enums.RepetitionB*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:REPetition
driver.configure.cell.carrier.hsupa.eagch.pattern.set_repetition(repetition =␣
↪enums.RepetitionB.CONTinuous)
```

Specifies whether the absolute grant pattern has to be transmitted only once, continuously or serving grant (SG) initialized. Select 'SG Initialized' only for E-RGCH measurements.

param repetition ONCE | CONTinuous | SGINit

Global Repeated Capabilities: repcap.Carrier

set_scope(*scope: List[bool]*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:SCOPe
driver.configure.cell.carrier.hsupa.eagch.pattern.set_scope(scope = [True,␣
↪False, True])
```

Specifies the absolute grant scopes of the absolute grant pattern. A query returns all eight defined scopes. A setting configures the first n scopes (n = 1 to 8) . Only the first m scopes are considered for transmission, with m specified via method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Eagch.Pattern.length.

param scope OFF | ON Comma-separated list of up to eight values OFF: absolute grant applies to all HARQ processes ON: absolute grant applies to one HARQ process only

Global Repeated Capabilities: repcap.Carrier

set_type_py(*type_py: List[bool]*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:TYPE
driver.configure.cell.carrier.hsupa.eagch.pattern.set_type_py(type_py = [True,␣
↪False, True])
```

Specifies the ID types of the absolute grant pattern. A query returns all eight defined types. A setting configures the first n types (n = 1 to 8) . Only the first m types are considered for transmission, with m specified via method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Eagch.Pattern.length.

param type_py OFF | ON Comma-separated list of up to eight values OFF: use primary UE-ID ON: use secondary UE-ID

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.12.1.2.4 Execute

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EAGCh:PATTern:EXECute
```

class Execute

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:EXECute
driver.configure.cell.carrier.hsupa.eagch.pattern.execute.set()
```

Triggers the execution of a single absolute grant pattern (repetition ONCE) .

Global Repeated Capabilities: repcap.Carrier

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EAGCh:PATTern:EXECute
driver.configure.cell.carrier.hsupa.eagch.pattern.execute.set_with_opc()
```

Triggers the execution of a single absolute grant pattern (repetition ONCE) .

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

Global Repeated Capabilities: repcap.Carrier

7.1.12.1.2.5 Ehich

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EHICh:MODE
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:EHICh:SIGNature
```

class Ehich

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

get_mode() → RsCmwWcdmaSig.enums.EhichIndicatorMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EHICh:MODE
value: enums.EhichIndicatorMode = driver.configure.cell.carrier.hsupa.ehich.get_
↪mode()
```

Specifies the HARQ acknowledgement indicator sequence transmitted via the E-HICH.

return mode: CRC | ALTernating | ACK | NACK | DTX CRC: react on UL CRC (ACK, NACK or DTX) ALTernating: alternating ACK, NACK ACK: all ACK NACK: all NACK DTX: all DTX

Global Repeated Capabilities: repcap.Carrier

get_signature() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EHICh:SIGNature
value: int = driver.configure.cell.carrier.hsupa.ehich.get_signature()
```

Specifies the E-HICH signature.

return signature: Range: 0 to 39

Global Repeated Capabilities: repcap.Carrier

set_mode(*mode: RsCmwWcdmaSig.enums.EhichIndicatorMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EHICh:MODE
driver.configure.cell.carrier.hsupa.ehich.set_mode(mode = enums.
↪EhichIndicatorMode.ACK)
```

Specifies the HARQ acknowledgement indicator sequence transmitted via the E-HICH.

param mode CRC | ALTernating | ACK | NACK | DTX CRC: react on UL CRC (ACK, NACK or DTX) ALTernating: alternating ACK, NACK ACK: all ACK NACK: all NACK DTX: all DTX

Global Repeated Capabilities: repcap.Carrier

`set_signature`(*signature: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:EHICh:SIGNature
driver.configure.cell.carrier.hsupa.ehich.set_signature(signature = 1)
```

Specifies the E-HICH signature.

param signature Range: 0 to 39

Global Repeated Capabilities: repcap.Carrier

7.1.12.1.2.6 Ergch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:ERGCh:MODE
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:ERGCh:SIGNature
```

class Ergch

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

`get_mode`() → RsCmwWcdmaSig.enums.ErgchIndicatorMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:MODE
value: enums.ErgchIndicatorMode = driver.configure.cell.carrier.hsupa.ergch.get_
↪mode()
```

Specifies the relative grant sequence transmitted via the E-RGCH. For definition of a user-defined pattern, see method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Ergch.Pattern.value.

return mode: ALTernating | HARQ | UP | DOWN | DTX | CONTinuous | SINGle ALTernating: alternating UP, DOWN - per TTI HARQ: alternating UP, DOWN - per HARQ cycle UP: all UP DOWN: all DOWN DTX: all DTX CONTinuous: continuous user-defined pattern SINGle: single user-defined pattern

Global Repeated Capabilities: repcap.Carrier

`get_signature`() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:SIGNature
value: int = driver.configure.cell.carrier.hsupa.ergch.get_signature()
```

Specifies the E-RGCH signature.

return signature: Range: 0 to 39

Global Repeated Capabilities: repcap.Carrier

set_mode(*mode: RsCmwWcdmaSig.enums.ErgchIndicatorMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:MODE
driver.configure.cell.carrier.hsupa.ergch.set_mode(mode = enums.
↪ErgchIndicatorMode.ALTernating)
```

Specifies the relative grant sequence transmitted via the E-RGCH. For definition of a user-defined pattern, see method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Ergch.Pattern.value.

param mode ALTernating | HARQ | UP | DOWN | DTX | CONTinuous | SINGle ALTernating: alternating UP, DOWN - per TTI HARQ: alternating UP, DOWN - per HARQ cycle UP: all UP DOWN: all DOWN DTX: all DTX CONTinuous: continuous user-defined pattern SINGle: single user-defined pattern

Global Repeated Capabilities: repcap.Carrier

set_signature(*signature: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:SIGNature
driver.configure.cell.carrier.hsupa.ergch.set_signature(signature = 1)
```

Specifies the E-RGCH signature.

param signature Range: 0 to 39

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.12.1.2.7 Pattern

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:ERGCh:PATTern:LENGth
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:ERGCh:PATTern
```

class Pattern

Pattern commands group definition. 3 total commands, 1 Sub-groups, 2 group commands

get_length() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:PATTern:LENGth
value: int = driver.configure.cell.carrier.hsupa.ergch.pattern.get_length()
```

Specifies the length of the user-defined relative grant pattern.

return length: Range: 1 to 8 (for 10 ms TTI: 1 to 4)

Global Repeated Capabilities: repcap.Carrier

get_value() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:PATTern
value: str = driver.configure.cell.carrier.hsupa.ergch.pattern.get_value()
```

Specifies the bits of the user-defined relative grant pattern. Bits exceeding the configured pattern length are ignored, see method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Ergch.Pattern.length.

return pattern: String containing exactly 8 bits 0 = DOWN, 1 = UP, - = DTX

Global Repeated Capabilities: repcap.Carrier

set_length(*length: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:PATTern:LENGth
driver.configure.cell.carrier.hsupa.ergch.pattern.set_length(length = 1)
```

Specifies the length of the user-defined relative grant pattern.

param length Range: 1 to 8 (for 10 ms TTI: 1 to 4)

Global Repeated Capabilities: repcap.Carrier

set_value(*pattern: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:PATTern
driver.configure.cell.carrier.hsupa.ergch.pattern.set_value(pattern = '1')
```

Specifies the bits of the user-defined relative grant pattern. Bits exceeding the configured pattern length are ignored, see method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Ergch.Pattern.length.

param pattern String containing exactly 8 bits 0 = DOWN, 1 = UP, - = DTX

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.12.1.2.8 Execute

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:ERGCh:PATTern:EXECute
```

class Execute

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:PATTern:EXECute
driver.configure.cell.carrier.hsupa.ergch.pattern.execute.set()
```

Triggers the execution of a single relative grant pattern (mode SINGle) .

Global Repeated Capabilities: repcap.Carrier

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ERGCh:PATTern:EXECute
driver.configure.cell.carrier.hsupa.ergch.pattern.execute.set_with_opc()
```

Triggers the execution of a single relative grant pattern (mode SINGle) .

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

Global Repeated Capabilities: repcap.Carrier

7.1.12.1.2.9 Etfci

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HSUPa:ETFCi:MSET
```

class Etfci

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

get_mset() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ETFCi:MSET
value: int or bool = driver.configure.cell.carrier.hsupa.etfci.get_mset()
```

Specifies the 'E-DCH minimum set E-TFCI' value signaled to the UE.

return min_set: Range: 0 to 127 Additional OFF | ON disables | enables the transmission of E-TFCI minimum set

Global Repeated Capabilities: repcap.Carrier

set_mset(*min_set: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>
↪:HSUPa:ETFCi:MSET
driver.configure.cell.carrier.hsupa.etfci.set_mset(min_set = 1)
```

Specifies the 'E-DCH minimum set E-TFCI' value signaled to the UE.

param min_set Range: 0 to 127 Additional OFF | ON disables | enables the transmission of E-TFCI minimum set

Global Repeated Capabilities: repcap.Carrier

7.1.12.1.3 Horder

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HORDer:DL
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CARRier<Carrier>:HORDer:UL
```

class Horder

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

get_downlink() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:HORDer:DL
value: bool = driver.configure.cell.carrier.horder.get_downlink()
```

Preconfigures the activation of an additional DL/UL carrier for the next HS-SCCH order type 1 in multi-carrier scenarios.

return enable: OFF | ON

Global Repeated Capabilities: repcap.Carrier

get_uplink() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:HORDer:UL
value: bool = driver.configure.cell.carrier.horder.get_uplink()
```

Preconfigures the activation of an additional DL/UL carrier for the next HS-SCCH order type 1 in multi-carrier scenarios.

return enable: OFF | ON

Global Repeated Capabilities: repcap.Carrier

set_downlink(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:HORDer:DL
driver.configure.cell.carrier.horder.set_downlink(enable = False)
```

Preconfigures the activation of an additional DL/UL carrier for the next HS-SCCH order type 1 in multi-carrier scenarios.

param enable OFF | ON

Global Repeated Capabilities: repcap.Carrier

set_uplink(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CARRier<carrier>:HORDer:UL
driver.configure.cell.carrier.horder.set_uplink(enable = False)
```

Preconfigures the activation of an additional DL/UL carrier for the next HS-SCCH order type 1 in multi-carrier scenarios.

param enable OFF | ON

Global Repeated Capabilities: repcap.Carrier

7.1.12.2 Rcause

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RCAuse:RRCRequest
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RCAuse:LOCation
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RCAuse:ATTach
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RCAuse:ROUTing
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RCAuse:CSRequest
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RCAuse:CSTYpe
```

class Rcause

Rcause commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

get_attach() → RsCmwWcdmaSig.enums.RejectionCauseB

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:ATTach
value: enums.RejectionCauseB = driver.configure.cell.rcause.get_attach()
```

Enables or disables the rejection of attach requests and selects the rejection cause to be transmitted.

return cause_number: C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C20 | C21 | C22 | C23 | C25 | C28 | C32 | C33 | C34 | C38 | C40 | C48 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13:

Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C40: No PDP context activated C48: retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional OFF | ON disables | enables the rejection of requests

get_cs_request() → RsCmwWcdmaSig.enums.RejectionCauseA

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:CSRequest
value: enums.RejectionCauseA = driver.configure.cell.rcause.get_cs_request()
```

Enables or disables the rejection of CM service requests and selects the rejection cause to be transmitted. The setting is relevant only for the specified service types, see method RsCmwWcdmaSig.Configure.Cell.Rcause.csType.

return cause_number: C2 | C3 | C6 | C11 | C12 | C13 | C15 | C96 | C99 | C100 | C111 | C4 | C5 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C97 | C98 | C101 C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters: OFF | ON (disables | enables the rejection of requests)

get_cs_type() → RsCmwWcdmaSig.enums.CmSerRejectType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:CSTYpe
value: enums.CmSerRejectType = driver.configure.cell.rcause.get_cs_type()
```

Specifies, to which type of CM service a request reject applies. Refer to method RsCmwWcdmaSig.Configure.Cell.Rcause. csRequest

return cm_ser_reject_type: NESMs | NCECall | NCSMs | ECSMs | NCALl | ECALl | SMS NESMs: Normal call + emergency call + SMS NCECall: Normal call + emergency call NCSMs: Normal call + SMS ECSMs: Emergency call + SMS NCALl: Normal call ECALl: Emergency call SMS: SMS

get_location() → RsCmwWcdmaSig.enums.RejectionCauseA

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:LOCation
value: enums.RejectionCauseA = driver.configure.cell.rcause.get_location()
```

Enables or disables the rejection of location update requests and selects the rejection cause to be transmitted.

return cause_number: C2 | C3 | C4 | C5 | C6 | C11 | C12 | C13 | C15 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional OFF | ON disables | enables the rejection of requests

get_routing() → RsCmwWcdmaSig.enums.RejectionCauseB

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:ROUTing
value: enums.RejectionCauseB = driver.configure.cell.rcause.get_routing()
```

Enables or disables the rejection of routing area update requests and selects the rejection cause to be transmitted.

return cause_number: C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C20 | C21 | C22 | C23 | C25 | C28 | C32 | C33 | C34 | C38 | C40 | C48 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C40: No PDP context activated C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

get_rrc_request() → RsCmwWcdmaSig.enums.RejectCause

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:RRCRequest
value: enums.RejectCause = driver.configure.cell.rcause.get_rrc_request()
```

Enables or disables the rejection of RRC connection requests and selects the rejection cause to be transmitted.

return reject_cause: CSCongestion | CSUNspecific | PSCongestion | PSUNspecific | ON | OFF CS/PS congestion, CS/PS unspecific reason Additional parameters: OFF | ON (disables | enables the rejection of requests)

set_attach(*cause_number: RsCmwWcdmaSig.enums.RejectionCauseB*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:ATTach
driver.configure.cell.rcause.set_attach(cause_number = enums.RejectionCauseB.
↪C10)
```

Enables or disables the rejection of attach requests and selects the rejection cause to be transmitted.

param cause_number C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C20 | C21 | C22 | C23 | C25 | C28 | C32 | C33 | C34 | C38 | C40 | C48 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C40: No PDP context activated C48: retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional OFF | ON disables | enables the rejection of requests

set_cs_request(*cause_number: RsCmwWcdmaSig.enums.RejectionCauseA*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:CSRequest
driver.configure.cell.rcause.set_cs_request(cause_number = enums.
↪RejectionCauseA.C100)
```

Enables or disables the rejection of CM service requests and selects the rejection cause to be transmitted. The setting is relevant only for the specified service types, see method RsCmwWcdmaSig.Configure.Cell.Rcause.csType.

param cause_number C2 | C3 | C6 | C11 | C12 | C13 | C15 | C96 | C99 | C100 | C111 | C4 | C5 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C97 | C98 | C101 C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in

location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters: OFF | ON (disables | enables the rejection of requests)

set_cs_type(*cm_ser_reject_type: RsCmwWcdmaSig.enums.CmSerRejectType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:CSTYpe
driver.configure.cell.rcause.set_cs_type(cm_ser_reject_type = enums.
↪CmSerRejectType.ECALl)
```

Specifies, to which type of CM service a request reject applies. Refer to method RsCmwWcdmaSig.Configure.Cell.Rcause. csRequest

param cm_ser_reject_type NESMs | NCECall | NCSMs | ECSMs | NCALl | ECALl | SMS NESMs: Normal call + emergency call + SMS NCECall: Normal call + emergency call NCSMs: Normal call + SMS ECSMs: Emergency call + SMS NCALl: Normal call ECALl: Emergency call SMS: SMS

set_location(*cause_number: RsCmwWcdmaSig.enums.RejectionCauseA*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:LOCation
driver.configure.cell.rcause.set_location(cause_number = enums.RejectionCauseA.
↪C100)
```

Enables or disables the rejection of location update requests and selects the rejection cause to be transmitted.

param cause_number C2 | C3 | C4 | C5 | C6 | C11 | C12 | C13 | C15 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional OFF | ON disables | enables the rejection of requests

set_routing(*cause_number: RsCmwWcdmaSig.enums.RejectionCauseB*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:ROUTing
driver.configure.cell.rcause.set_routing(cause_number = enums.RejectionCauseB.
↪C10)
```

Enables or disables the rejection of routing area update requests and selects the rejection cause to be transmitted.

param cause_number C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C20 | C21 | C22 | C23 | C25 | C28 | C32 | C33 | C34 | C38 | C40 | C48 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C40: No PDP context activated C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

set_rrc_request(*reject_cause: RsCmwWcdmaSig.enums.RejectCause*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RCAuse:RRCRequest
driver.configure.cell.rcause.set_rrc_request(reject_cause = enums.RejectCause.
↪CSCongestion)
```

Enables or disables the rejection of RRC connection requests and selects the rejection cause to be transmitted.

param reject_cause CSCongestion | CSUNspecific | PSCongestion | PSUNspecific | ON | OFF CS/PS congestion, CS/PS unspecific reason Additional parameters: OFF | ON (disables | enables the rejection of requests)

7.1.12.3 Timeout

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TOUT:MOC
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TOUT:ATOFfset
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TOUT:PPIF
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TOUT:PREPetitions
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TOUT:OSYNch
```

class Timeout

Timeout commands group definition. 7 total commands, 2 Sub-groups, 5 group commands

get_at_offset() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:ATOFfset
value: int = driver.configure.cell.timeout.get_at_offset()
```

Specifies a delay value, used by the RRC for calculation of the activation time in peer messages. Low values correspond to fast signaling, high values to slow signaling.

return offset: Range: 0 to 10

get_moc() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:MOC
value: int = driver.configure.cell.timeout.get_moc()
```

Defines the time period of R&S CMW alerting state.

return timeout: 0: the alerting state is skipped 1 to 255: time period the R&S CMW waits before changes to 'Call Established' state Range: 0 to 255, Unit: s

get_osynch() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:OSYNch
value: int = driver.configure.cell.timeout.get_osynch()
```

Sets the out-of-synchronization timeout value. This value specifies the time after which the instrument, having waited for a signal from the connected UE, releases the connection and returns to state registered.

return value: Range: 2 s to 25 s, Unit: s

get_ppif() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:PPIF
value: int = driver.configure.cell.timeout.get_ppif()
```

Number of paging indicators that the R&S CMW transmits in each PICH frame.

return indications: 18 | 36 | 72 | 144

get_prepetitions() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:PREPetitions
value: int = driver.configure.cell.timeout.get_prepetitions()
```

Specifies the number of paging procedures to be performed if the UE does not answer paging.

return repetitions: Range: 0 to 65535

set_at_offset(*offset: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:ATOFfset
driver.configure.cell.timeout.set_at_offset(offset = 1)
```

Specifies a delay value, used by the RRC for calculation of the activation time in peer messages. Low values correspond to fast signaling, high values to slow signaling.

param offset Range: 0 to 10

set_moc(*timeout: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:MOC
driver.configure.cell.timeout.set_moc(timeout = 1)
```

Defines the time period of R&S CMW alerting state.

param timeout 0: the alerting state is skipped 1 to 255: time period the R&S CMW waits before changes to 'Call Established' state Range: 0 to 255, Unit: s

set_osynch(*value: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:OSYNch
driver.configure.cell.timeout.set_osynch(value = 1)
```

Sets the out-of-synchronization timeout value. This value specifies the time after which the instrument, having waited for a signal from the connected UE, releases the connection and returns to state registered.

param value Range: 2 s to 25 s, Unit: s

set_ppif(*indications: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:PPIF
driver.configure.cell.timeout.set_ppif(indications = 1)
```

Number of paging indicators that the R&S CMW transmits in each PICH frame.

param indications 18 | 36 | 72 | 144

set_prepetitions(*repetitions: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:PREPetitions
driver.configure.cell.timeout.set_prepetitions(repetitions = 1)
```

Specifies the number of paging procedures to be performed if the UE does not answer paging.

param repetitions Range: 0 to 65535

Cloning the Group

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

Subgroups

7.1.12.3.1 N<CounterNo>

RepCap Settings

```
# Range: Nr313 .. Nr313
rc = driver.configure.cell.timeout.n.repcap_counterNo_get()
driver.configure.cell.timeout.n.repcap_counterNo_set(repcap.CounterNo.Nr313)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TOUT:N<CounterNo>
```

class N

N commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: CounterNo, default value after init: CounterNo.Nr313

get(*counterNo=<CounterNo.Default: -1>*) → RsCmwWcdmaSig.enums.CounterValue

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:N<nr>
value: enums.CounterValue = driver.configure.cell.timeout.n.get(counterNo =␣
↪repcap.CounterNo.Default)
```

Sets a maximum value for counter N313. The UE counts successive 'out of sync' indications received from layer 1. When the maximum value is reached, the UE considers a 'radio link failure' condition and a connection release.

param counterNo optional repeated capability selector. Default value: Nr313 (settable in the interface 'N')

return value: N1 | N2 | N4 | N10 | N20 | N50 | N100 | N200 Maximum counter value prefixed by N.

set(*value: RsCmwWcdmaSig.enums.CounterValue*, *counterNo=<CounterNo.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:N<nr>
driver.configure.cell.timeout.n.set(value = enums.CounterValue.N1, counterNo =␣
↪repcap.CounterNo.Default)
```

Sets a maximum value for counter N313. The UE counts successive 'out of sync' indications received from layer 1. When the maximum value is reached, the UE considers a 'radio link failure' condition and a connection release.

param value N1 | N2 | N4 | N10 | N20 | N50 | N100 | N200 Maximum counter value prefixed by N.

param counterNo optional repeated capability selector. Default value: Nr313 (settable in the interface 'N')

Cloning the Group

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

7.1.12.3.2 T<Timer>

RepCap Settings

```
# Range: T313 .. T3312
rc = driver.configure.cell.timeout.t.repcap_timer_get()
driver.configure.cell.timeout.t.repcap_timer_set(repcap.Timer.T313)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TOUT:T<Timer>
```

class T

T commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: Timer, default value after init: Timer.T313

get(*timer=<Timer.Default: -1>*) → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:T<nr>
value: int = driver.configure.cell.timeout.t.get(timer = repcap.Timer.Default)
```

Set the timeout value for timer T3212 and T3312.

param timer optional repeated capability selector. Default value: T313 (settable in the interface 'T')

return value: Range: 0 to 255, Unit: 6 minutes for T3212, 2 seconds for T3312

set(*value: int*, *timer=<Timer.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TOUT:T<nr>
driver.configure.cell.timeout.t.set(value = 1, timer = repcap.Timer.Default)
```

Set the timeout value for timer T3212 and T3312.

param value Range: 0 to 255, Unit: 6 minutes for T3212, 2 seconds for T3312

param timer optional repeated capability selector. Default value: T313 (settable in the interface 'T')

Cloning the Group

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

7.1.12.4 Request

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:REQuest:IMEI
CONFigure:WCDMa:SIGNaling<Instance>:CELL:REQuest:ADETach
CONFigure:WCDMa:SIGNaling<Instance>:CELL:REQuest:RCUR
```

class Request

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

get_adetach() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:REQuest:ADETach
value: bool = driver.configure.cell.request.get_adetach()
```

Enables or disables the CS registration and PS attach procedure.

return enable: OFF | ON

get_imei() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:REQuest:IMEI
value: bool = driver.configure.cell.request.get_imei()
```

Enables or disables the request of the IMEI from the UE.

return enable: OFF | ON

get_rcur() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:REQuest:RCUR
value: bool = driver.configure.cell.request.get_rcur()
```

Enables or disables the request of the radio capability update from the UE.

return enable: OFF | ON

set_adetach(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:REQuest:ADETach
driver.configure.cell.request.set_adetach(enable = False)
```

Enables or disables the CS registration and PS attach procedure.

param enable OFF | ON

set_imei(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:REQuest:IMEI
driver.configure.cell.request.set_imei(enable = False)
```

Enables or disables the request of the IMEI from the UE.

param enable OFF | ON

set_rcur(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:REQuest:RCUR
driver.configure.cell.request.set_rcur(enable = False)
```

Enables or disables the request of the radio capability update from the UE.

param enable OFF | ON

7.1.12.5 Security

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SECurity:CIPHering
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SECurity:OPC
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SECurity:SIMCard
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SECurity:SKEY
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SECurity:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SECurity:AUTHenticat
```

class Security

Security commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

get_authenticate() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:AUTHenticat
value: bool = driver.configure.cell.security.get_authenticate()
```

Enables or disables authentication, to be performed during registration.

return enable: OFF | ON

get_ciphering() → RsCmwWcdmaSig.enums.Cipher

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:CIPHering
value: enums.Cipher = driver.configure.cell.security.get_ciphering()
```

Specifies ciphering to be used for a radio bearer.

return cipher: UEA0 | UEA1 | UEA2 UEA0: no ciphering UEA1: algorithm 1 (KASUMI) UEA2: algorithm 2 (SNOW 3G)

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:ENABle
value: bool = driver.configure.cell.security.get_enable()
```

Enables or disables the security mode during authentication. With enabled security mode, the UE performs an integrity check.

return enable: OFF | ON

get_opc() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:OPC
value: float = driver.configure.cell.security.get_opc()
```

Specifies the key OPc as 32-digit hexadecimal number.

return opc: Range: #H00000000000000000000000000000000 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

get_sim_card() → RsCmwWcdmaSig.enums.SimCardType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:SIMCard
value: enums.SimCardType = driver.configure.cell.security.get_sim_card()
```

Selects the type of the SIM card used for registration.

return sim_card_type: C3G | C2G | MILenage C3G: 3G USIM C2G: 2G SIM MILenage: USIM with MILENAGE algorithm set

get_skey() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:SKEY
value: float = driver.configure.cell.security.get_skey()
```

Defines the secret key K as 32-digit hexadecimal number. Leading zeros can be omitted. K is used for the authentication procedure including a possible integrity check.

return secret_key: Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

set_authenticate(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:AUTHenticat
driver.configure.cell.security.set_authenticate(enable = False)
```

Enables or disables authentication, to be performed during registration.

param enable OFF | ON

set_ciphering(*cipher: RsCmwWcdmaSig.enums.Cipher*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:CIPHering
driver.configure.cell.security.set_ciphering(cipher = enums.Cipher.UEA0)
```

Specifies ciphering to be used for a radio bearer.

param cipher UEA0 | UEA1 | UEA2 UEA0: no ciphering UEA1: algorithm 1 (KASUMI) UEA2: algorithm 2 (SNOW 3G)

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:ENABle
driver.configure.cell.security.set_enable(enable = False)
```

Enables or disables the security mode during authentication. With enabled security mode, the UE performs an integrity check.

param enable OFF | ON

set_opc(*opc: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:OPC
driver.configure.cell.security.set_opc(opc = 1.0)
```

Specifies the key OPc as 32-digit hexadecimal number.

param opc Range: #H00000000000000000000000000000000 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

set_sim_card(*sim_card_type: RsCmwWcdmaSig.enums.SimCardType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:SIMCard
driver.configure.cell.security.set_sim_card(sim_card_type = enums.SimCardType.
↪C2G)
```

Selects the type of the SIM card used for registration.

param sim_card_type C3G | C2G | MILenage C3G: 3G USIM C2G: 2G SIM MILenage: USIM with MILENAGE algorithm set

set_skey(*secret_key: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SECurity:SKEY
driver.configure.cell.security.set_skey(secret_key = 1.0)
```

Defines the secret key K as 32-digit hexadecimal number. Leading zeros can be omitted. K is used for the authentication procedure including a possible integrity check.

param secret_key Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

7.1.12.6 UeIdentity

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:UEIDentity:FILTer
CONFigure:WCDMa:SIGNaling<Instance>:CELL:UEIDentity:IMSI
CONFigure:WCDMa:SIGNaling<Instance>:CELL:UEIDentity:USE
```

class UeIdentity

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

get_filter_py() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:UEIDentity:FILTer
value: bool = driver.configure.cell.ueIdentity.get_filter_py()
```

If enabled, the R&S CMW allows only the default IMSI to execute location update and attach.

return enable: OFF | ON

get_imsi() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:UEIDentity:IMSI
value: str = driver.configure.cell.ueIdentity.get_imsi()
```

Specifies the default IMSI that the instrument can use before the UE is registered.

return value: String value, containing 15 digits.

get_use() → RsCmwWcdmaSig.enums.Enable

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:UEIDentity:USE
value: enums.Enable = driver.configure.cell.ueIdentity.get_use()
```

Specifies whether the default IMSI is used. The default IMSI is defined via method RsCmwWcdmaSig.Configure.Cell. UeIdentity.imsi. You can only enable the default IMSI but not disable it. Instead it is disabled automatically when registration is performed with a different IMSI.

return enable: ON

set_filter_py(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:UEIDentity:FILTer
driver.configure.cell.ueIdentity.set_filter_py(enable = False)
```

If enabled, the R&S CMW allows only the default IMSI to execute location update and attach.

param enable OFF | ON

set_imsi(*value: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:UEIDentity:IMSI
driver.configure.cell.ueIdentity.set_imsi(value = '1')
```

Specifies the default IMSI that the instrument can use before the UE is registered.

param value String value, containing 15 digits.

set_use(*enable: RsCmwWcdmaSig.enums.Enable*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:UEIDentity:USE
driver.configure.cell.ueIdentity.set_use(enable = enums.Enable.ON)
```

Specifies whether the default IMSI is used. The default IMSI is defined via method RsCmwWcdmaSig.Configure.Cell. UeIdentity.imsi. You can only enable the default IMSI but not disable it. Instead it is disabled automatically when registration is performed with a different IMSI.

param enable ON

7.1.12.7 Mnc

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:MNC:DIGits
CONFigure:WCDMa:SIGNaling<Instance>:CELL:MNC
```

class Mnc

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

class ValueStruct

Structure for reading output parameters. Fields:

- Value: int: Range: 0 to 99 or 999 depending on NrOfDigits
- Nr_Of_Digits: enums.NrOfDigits: D2 | D3 D2: two-digit MNC D3: three-digit MNC

get_digits() → RsCmwWcdmaSig.enums.NrOfDigits

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MNC:DIGits
value: enums.NrOfDigits = driver.configure.cell.mnc.get_digits()
```

Specifies the size of mobile network code (MNC) . A two or three-digit MNC can be selected.

return no_digits: D2 | D3 D2: two-digit MNC D3: three-digit MNC

get_value() → ValueStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MNC
value: ValueStruct = driver.configure.cell.mnc.get_value()
```

Specifies the mobile network code (MNC) . A two or three-digit MNC can be set. Leading zeros can be omitted.

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

set_digits(*no_digits: RsCmwWcdmaSig.enums.NrOfDigits*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MNC:DIGits
driver.configure.cell.mnc.set_digits(no_digits = enums.NrOfDigits.D2)
```

Specifies the size of mobile network code (MNC) . A two or three-digit MNC can be selected.

param no_digits D2 | D3 D2: two-digit MNC D3: three-digit MNC

set_value(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Mnc.Mnc.ValueStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:MNC
driver.configure.cell.mnc.set_value(value = ValueStruct())
```

Specifies the mobile network code (MNC) . A two or three-digit MNC can be set. Leading zeros can be omitted.

param value see the help for ValueStruct structure arguments.

7.1.12.8 ReSelection

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RESelection:SEARch
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RESelection:QUALity
CONFigure:WCDMa:SIGNaling<Instance>:CELL:RESelection:TIME
```

class ReSelection

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

class QualityStruct

Structure for reading output parameters. Fields:

- Qqual_Min: float: Minimum required quality level in the reselection target cell. Range: -24 dB to 0 dB, Unit: dB
- Qrxlevmin: float: Minimum RX level at a UE antenna required for reselection to the UMTS cell Range: -115 dBm to -25 dBm, Unit: dBm
- Qrx_Lev_Min_Eutra: float: Minimum RX level at a UE antenna required for access to the LTE cell Range: -140 dBm to -44 dBm, Unit: dBm
- Qhyst_1_S: float: Hysteresis used for GSM, TDD and for FDD cells in case the quality measure for reselection is set to CPICH RSCP Range: 0 dB to 40 dB, Unit: dB
- Qhyst_2_S: float: Hysteresis used for FDD cells if the quality measure for reselection is set to CPICH Ec/No Range: 0 dB to 40 dB, Unit: dB

class SearchStruct

Structure for reading output parameters. Fields:

- Sintra_Search: float: Range: -32 dB to 20 dB, Unit: dB
- Sinter_Search: float: Range: -32 dB to 20 dB, Unit: dB
- Ssearch_Rat: float: Range: -32 dB to 20 dB, Unit: dB

get_quality() → QualityStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RESelection:QUALity
value: QualityStruct = driver.configure.cell.reSelection.get_quality()
```

Defines the power levels required for cell reselection. They are transmitted to the UE in the system information.

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

get_search() → SearchStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RESelection:SEARch
value: SearchStruct = driver.configure.cell.reSelection.get_search()
```

Defines the thresholds Sintrasearch, Sintersearch and S searchRAT m = GSM required for cell reselection. They are transmitted to the UE in the system information.

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

get_time() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RESelection:TIME
value: float = driver.configure.cell.reSelection.get_time()
```

Sets the time hysteresis for the cell reselection algorithm.

return tre_selections: Range: 0 s to 31 s

set_quality(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.ReSelection.ReSelection.QualityStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RESelection:QUALity
driver.configure.cell.reSelection.set_quality(value = QualityStruct())
```

Defines the power levels required for cell reselection. They are transmitted to the UE in the system information.

param value see the help for QualityStruct structure arguments.

set_search(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.ReSelection.ReSelection.SearchStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RESelection:SEARch
driver.configure.cell.reSelection.set_search(value = SearchStruct())
```

Defines the thresholds Sintrasearch, Sintersearch and S searchRAT m = GSM required for cell reselection. They are transmitted to the UE in the system information.

param value see the help for SearchStruct structure arguments.

set_time(*tre_selections: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:RESelection:TIME
driver.configure.cell.reSelection.set_time(tre_selections = 1.0)
```

Sets the time hysteresis for the cell reselection algorithm.

param tre_selections Range: 0 s to 31 s

7.1.12.9 Time

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:TSOurce
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:DATE
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:TIME
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:DSTime
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:LTZoffset
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:SREGister
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:SNName
```

class Time

Time commands group definition. 8 total commands, 1 Sub-groups, 7 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_date() → DateStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:DATE
value: DateStruct = driver.configure.cell.time.get_date()
```

Specifies the UTC date for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time.tsource) .

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

get_daylight_saving_time() → RsCmwWcdmaSig.enums.DsTime

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:DSTime
value: enums.DsTime = driver.configure.cell.time.get_daylight_saving_time()
```

Specifies a daylight saving time (DST) offset for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time. tsource) .

return enable: P1H | P2H | ON | OFF P1H: +1h offset P2H: +2h offset Additional OFF | ON disables | enables DST

get_ltz_offset() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:LTZoffset
value: float = driver.configure.cell.time.get_ltz_offset()
```

Specifies the time zone offset for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time.tsource) .

return time_zone_offset: Range: -19.75 h to 19.75 h, Unit: h

get_snname() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:SNName
value: bool = driver.configure.cell.time.get_snname()
```

If enabled, sends the full and short network name within date and time signaling to the UE.

return enable: OFF | ON

get_sregister() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:SREGister
value: bool = driver.configure.cell.time.get_sregister()
```

Specifies whether the date and time information is sent to the UE during the registration and attach procedure or not.

return enable: OFF | ON ON: send date and time at registration/attach OFF: do not send date and time at registration/attach

get_time() → TimeStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:TIME
value: TimeStruct = driver.configure.cell.time.get_time()
```

Specifies the UTC time for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time.tsource) .

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

get_tsource() → RsCmwWcdmaSig.enums.SourceTime

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:TSOurce
value: enums.SourceTime = driver.configure.cell.time.get_tsource()
```

Selects the date and time source. INTRO_CMD_HELP: The time source DATE is configured via the following commands:

- method RsCmwWcdmaSig.Configure.Cell.Time.date
- method RsCmwWcdmaSig.Configure.Cell.Time.time
- method RsCmwWcdmaSig.Configure.Cell.Time.daylightSavingTime

return source_time: CMWTime | DATE CMWTime: Windows date and time DATE: Date and time specified via remote commands

set_date(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Time.Time.DateStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:DATE
driver.configure.cell.time.set_date(value = DateStruct())
```

Specifies the UTC date for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time.tsource) .

param value see the help for DateStruct structure arguments.

set_daylight_saving_time(*enable: RsCmwWcdmaSig.enums.DsTime*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:DSTime
driver.configure.cell.time.set_daylight_saving_time(enable = enums.DsTime.OFF)
```

Specifies a daylight saving time (DST) offset for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time. tsource) .

param enable P1H | P2H | ON | OFF P1H: +1h offset P2H: +2h offset Additional OFF | ON disables | enables DST

set_ltz_offset(*time_zone_offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:LTZoffset
driver.configure.cell.time.set_ltz_offset(time_zone_offset = 1.0)
```

Specifies the time zone offset for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time.tsource) .

param time_zone_offset Range: -19.75 h to 19.75 h, Unit: h

set_snname(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:SNName
driver.configure.cell.time.set_snname(enable = False)
```

If enabled, sends the full and short network name within date and time signaling to the UE.

param enable OFF | ON

set_sregister(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:SREGister
driver.configure.cell.time.set_sregister(enable = False)
```

Specifies whether the date and time information is sent to the UE during the registration and attach procedure or not.

param enable OFF | ON ON: send date and time at registration/attach OFF: do not send date and time at registration/attach

set_time(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Time.Time.TimeStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:TIME
driver.configure.cell.time.set_time(value = TimeStruct())
```

Specifies the UTC time for the time source DATE (see method RsCmwWcdmaSig.Configure.Cell.Time.tsource) .

param value see the help for TimeStruct structure arguments.

set_tsource(*source_time: RsCmwWcdmaSig.enums.SourceTime*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:TSOurce
driver.configure.cell.time.set_tsource(source_time = enums.SourceTime.CMWTime)
```

Selects the date and time source. INTRO_CMD_HELP: The time source DATE is configured via the following commands:

- method RsCmwWcdmaSig.Configure.Cell.Time.date
- method RsCmwWcdmaSig.Configure.Cell.Time.time

- method RsCmwWcdmaSig.Configure.Cell.Time.daylightSavingTime

  param source_time CMWTime | DATE CMWTime: Windows date and time DATE: Date and time specified via remote commands

Cloning the Group

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

Subgroups

7.1.12.9.1 Snow

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:TIME:SNOW
```

class Snow

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:SNOW
driver.configure.cell.time.snow.set()
```

Triggers the transfer of the date and time information to the UE.

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:TIME:SNOW
driver.configure.cell.time.snow.set_with_opc()
```

Triggers the transfer of the date and time information to the UE.

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

7.1.12.10 Sync

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SYNC:ZONE
CONFigure:WCDMa:SIGNaling<Instance>:CELL:SYNC:OFFSet
```

class Sync

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

get_offset() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SYNC:OFFSet
value: float = driver.configure.cell.sync.get_offset()
```

Configures the timing offset relative to the time zone.

return offset: Range: -38399 chips / -99997E-5 s to 0 chips / 0 s, Unit: s

get_zone() → RsCmwWcdmaSig.enums.Zone

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SYNC:ZONE
value: enums.Zone = driver.configure.cell.sync.get_zone()
```

Selects the synchronization zone for the signaling application.

return zone: NONE | Z1 NONE: no synchronization Z1: synchronization to zone 1

set_offset(*offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SYNC:OFFSet
driver.configure.cell.sync.set_offset(offset = 1.0)
```

Configures the timing offset relative to the time zone.

param offset Range: -38399 chips / -99997E-5 s to 0 chips / 0 s, Unit: s

set_zone(*zone: RsCmwWcdmaSig.enums.Zone*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:SYNC:ZONE
driver.configure.cell.sync.set_zone(zone = enums.Zone.NONE)
```

Selects the synchronization zone for the signaling application.

param zone NONE | Z1 NONE: no synchronization Z1: synchronization to zone 1

7.1.12.11 Hsdpa

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:ANRFactor
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:TYPE
```

class Hsdpa

Hsdpa commands group definition. 23 total commands, 4 Sub-groups, 2 group commands

get_anr_factor() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:ANRFactor
value: int = driver.configure.cell.hsdpa.get_anr_factor()
```

Specifies the number of transmissions of the same ACK/NACK (ACK/NACK repetition factor) .

return factor: Range: 1 to 4

get_type_py() → RsCmwWcdmaSig.enums.ChannelType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:TYPE
value: enums.ChannelType = driver.configure.cell.hsdpa.get_type_py()
```

Selects the configuration type of the high-speed downlink shared channel (HS-DSCH) .

return channel_type: FIXed | CQI | UDEFined FIXed: fixed reference channel CQI: channel for CQI reporting tests UDEFined: user-defined channel configuration

set_anr_factor(*factor: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:ANRFactor
driver.configure.cell.hsdpa.set_anr_factor(factor = 1)
```

Specifies the number of transmissions of the same ACK/NACK (ACK/NACK repetition factor) .

param factor Range: 1 to 4

set_type_py(*channel_type: RsCmwWcdmaSig.enums.ChannelType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:TYPE
driver.configure.cell.hsdpa.set_type_py(channel_type = enums.ChannelType.CQI)
```

Selects the configuration type of the high-speed downlink shared channel (HS-DSCH) .

param channel_type FIXed | CQI | UDEFined FIXed: fixed reference channel CQI: channel for CQI reporting tests UDEFined: user-defined channel configuration

Cloning the Group

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

Subgroups

7.1.12.11.1 UeCategory

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UECategory:MANual
```

class UeCategory

UeCategory commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get_manual() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UECategory:MANual
value: int = driver.configure.cell.hsdpa.ueCategory.get_manual()
```

Configures the UE category to be used by the R&S CMW if no reported value is available or usage of the reported value is disabled, see method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UeCategory.Reported.set.

return ue_cat_manual: Range: 1 to 24, 29 to 32

set_manual(*ue_cat_manual: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UECategory:MANual
driver.configure.cell.hsdpa.ueCategory.set_manual(ue_cat_manual = 1)
```

Configures the UE category to be used by the R&S CMW if no reported value is available or usage of the reported value is disabled, see method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UeCategory.Reported.set.

param ue_cat_manual Range: 1 to 24, 29 to 32

Cloning the Group

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

Subgroups

7.1.12.11.1.1 Reported

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UECategory:REPorted
```

class Reported

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

class GetStruct

Response structure. Fields:

- Use_Reported: bool: OFF | ON
- Ue_Cat_Reported: int: UE category reported by the UE (NAV indicates that none has been reported) Range: 1 to 24

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UECategory:REPorted
value: GetStruct = driver.configure.cell.hsdpa.ueCategory.reported.get()
```

Enable or disable usage of the UE category value reported by the UE. When disabled, the UE category must be set manually, see method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UeCategory.manual. The manually set value is also used if no reported value is available.

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

set(*use_reported: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UECategory:REPorted
driver.configure.cell.hsdpa.ueCategory.reported.set(use_reported = False)
```

Enable or disable usage of the UE category value reported by the UE. When disabled, the UE category must be set manually, see method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UeCategory.manual. The manually set value is also used if no reported value is available.

param use_reported OFF | ON

7.1.12.11.2 Fixed

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:FIXed:HSET
```

class Fixed

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

get_hset() → RsCmwWcdmaSig.enums.HsetFixed

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:FIXed:HSET
value: enums.HsetFixed = driver.configure.cell.hsdpa.fixed.get_hset()
```

Selects an H-Set for the fixed reference channel.

return hset: H1M1 | H1M2 | H2M1 | H2M2 | H3M1 | H3M2 | H4M1 | H5M1 | H6M1 | H6M2 | H8M3 | H8MT | H1MI | H8MI | H3A1 | H3A2 | H8A3 | H8AI | HAM1 | HAM2 | HAA1 | HAA2 | HCM1 | HCMT | H6A1 | H6A2 | H1AI | H1BI | H3B1 | H3B2 | H6B1 | H6B2 | H8B3 | H8BI | HAB1 | HAB2 Single carrier H-Sets: H1M1 to H6M1, HAM1: H-Set 1 to 6, 10 (QPSK) H1M2 to H3M2, H6M2, HAM2: H-Set 1 to 3, 6, 10 (16-QAM) H8M3: H-Set 8 (64-QAM) H1MI, H8MI: H-Set 1, 8 (maximum input) H8MT: H-Set 8 (maximum throughput) Dual carrier H-Sets: H1AI: H-Set 1A (maximum input) H3A1, H6A1, HAA1, HCM1: H-Set 3A, 6A, 10A, 12 (QPSK) H3A2, H6A2, HAA2: H-Set 3A, 6A, 10A (16-QAM) H8A3: H-Set 8A (64-QAM) H8AI: H-Set 8A (maximum input) HCMT: H-Set 12 (maximum throughput) 3C-HSDPA H-Sets: H1BI: H-Set 1B (maximum input) H3B1, H6B1, HAB1: H-Set 3B, 6B, 10B (QPSK) H3B2, H6B2, HAB2: H-Set 3B, 6B, 10B (16-QAM) H8B3: H-Set 8B (64-QAM) H8BI: H-Set 8B (maximum input)

set_hset(*hset: RsCmwWcdmaSig.enums.HsetFixed*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:FIXed:HSET
driver.configure.cell.hsdpa.fixed.set_hset(hset = enums.HsetFixed.H1AI)
```

Selects an H-Set for the fixed reference channel.

param hset H1M1 | H1M2 | H2M1 | H2M2 | H3M1 | H3M2 | H4M1 | H5M1 | H6M1 | H6M2 | H8M3 | H8MT | H1MI | H8MI | H3A1 | H3A2 | H8A3 | H8AI | HAM1 | HAM2 | HAA1 | HAA2 | HCM1 | HCMT | H6A1 | H6A2 | H1AI | H1BI | H3B1 | H3B2 | H6B1 | H6B2 | H8B3 | H8BI | HAB1 | HAB2 Single carrier H-Sets: H1M1 to H6M1, HAM1: H-Set 1 to 6, 10 (QPSK) H1M2 to H3M2, H6M2, HAM2: H-Set 1 to 3, 6, 10 (16-QAM) H8M3: H-Set 8 (64-QAM) H1MI, H8MI: H-Set 1, 8 (maximum input) H8MT: H-Set 8 (maximum throughput) Dual carrier H-Sets: H1AI:

H-Set 1A (maximum input) H3A1, H6A1, HAA1, HCM1: H-Set 3A, 6A, 10A, 12 (QPSK) H3A2, H6A2, HAA2: H-Set 3A, 6A, 10A (16-QAM) H8A3: H-Set 8A (64-QAM) H8AI: H-Set 8A (maximum input) HCMT: H-Set 12 (maximum throughput) 3C-HSDPA H-Sets: H1BI: H-Set 1B (maximum input) H3B1, H6B1, HAB1: H-Set 3B, 6B, 10B (QPSK) H3B2, H6B2, HAB2: H-Set 3B, 6B, 10B (16-QAM) H8B3: H-Set 8B (64-QAM) H8BI: H-Set 8B (maximum input)

7.1.12.11.3 Cqi

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:RFACtor
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:FBCYcle
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:TTI
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:HARQ
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:TINDex
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:SEQuence
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:FOLLow
```

class Cqi

Cqi commands group definition. 12 total commands, 2 Sub-groups, 7 group commands

class FollowStruct

Structure for reading output parameters. Fields:

- Min_Value: int: Range: 1 to 30
- Max_Value: int: Range: 1 to 30

class SequenceStruct

Structure for reading output parameters. Fields:

- Min_Value: int: Range: 1 to 30
- Max_Value: int: Range: 1 to 30

get_fb_cycle() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:FBCYcle
value: float or bool = driver.configure.cell.hsdpa.cqi.get_fb_cycle()
```

Specifies the time after which the UE sends a new CQI value on the HS-DPCCH (CQI feedback cycle) . The CQI transmission can also be disabled completely.

return feedback_cycle: Range: 2 ms to 160 ms, Unit: s Additional parameters: OFF | ON (disables | enables CQI transmission)

get_follow() → FollowStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:FOLLow
value: FollowStruct = driver.configure.cell.hsdpa.cqi.get_follow()
```

Defines the allowed range of CQI table indices. A value proposed by the UE is accepted if it is located within the range and FOLLow is configured via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.tindex.

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

get_harq() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:HARQ
value: int = driver.configure.cell.hsdpa.cqi.get_harq()
```

Specifies the number of HARQ processes.

return number: Range: 1 to 8

get_rfactor() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RFACtor
value: int = driver.configure.cell.hsdpa.cqi.get_rfactor()
```

Specifies how often the UE transmits the same CQI value per feedback cycle (CQI repetition factor) .

return factor: Range: 1 to 4

get_sequence() → SequenceStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:SEQuence
value: SequenceStruct = driver.configure.cell.hsdpa.cqi.get_sequence()
```

Selects the range of CQI table indices to be used cyclically if SEQuence is configured via method RsCmwWcdmaSig.Configure. Cell.Hsdpa.Cqi.tindex.

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

get_tindex() → RsCmwWcdmaSig.enums.TableIndex

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:TINDex
value: enums.TableIndex = driver.configure.cell.hsdpa.cqi.get_tindex()
```

Specifies the method to be used for selection of the CQI table index.

return table_index: FIXed | SEQuence | CONFormance | FOLLow FIXed A fixed mapping table row is used. See also method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsdpa.Cqi.fixed SEQuence A sequence of mapping table rows is used. See also method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.sequence CONFormance A CQI reporting test is to be performed. See also method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsdpa.Cqi.conformance FOLLow The CQI value to be used is proposed by the UE. See also method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.follow

get_tti() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:TTI
value: int = driver.configure.cell.hsdpa.cqi.get_tti()
```

Queries the minimum distance between two consecutive transmission time intervals in which the HS-DSCH is allocated to the UE.

return tti: Range: 1 to 3

set_fb_cycle(*feedback_cycle: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:FBCYcle
driver.configure.cell.hsdpa.cqi.set_fb_cycle(feedback_cycle = 1.0)
```

Specifies the time after which the UE sends a new CQI value on the HS-DPCCH (CQI feedback cycle) . The CQI transmission can also be disabled completely.

param feedback_cycle Range: 2 ms to 160 ms, Unit: s Additional parameters: OFF | ON (disables | enables CQI transmission)

set_follow(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Hsdpa_.Cqi.Cqi.FollowStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:FOLLow
driver.configure.cell.hsdpa.cqi.set_follow(value = FollowStruct())
```

Defines the allowed range of CQI table indices. A value proposed by the UE is accepted if it is located within the range and FOLLow is configured via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.tindex.

param value see the help for FollowStruct structure arguments.

set_harq(*number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:HARQ
driver.configure.cell.hsdpa.cqi.set_harq(number = 1)
```

Specifies the number of HARQ processes.

param number Range: 1 to 8

set_rfactor(*factor: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RFACtor
driver.configure.cell.hsdpa.cqi.set_rfactor(factor = 1)
```

Specifies how often the UE transmits the same CQI value per feedback cycle (CQI repetition factor) .

param factor Range: 1 to 4

set_sequence(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Hsdpa_.Cqi.Cqi.SequenceStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:SEQuence
driver.configure.cell.hsdpa.cqi.set_sequence(value = SequenceStruct())
```

Selects the range of CQI table indices to be used cyclically if SEQuence is configured via method RsCmwWcdmaSig.Configure. Cell.Hsdpa.Cqi.tindex.

param value see the help for SequenceStruct structure arguments.

set_tindex(*table_index: RsCmwWcdmaSig.enums.TableIndex*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:TINDex
driver.configure.cell.hsdpa.cqi.set_tindex(table_index = enums.TableIndex.
↪CONFormance)
```

Specifies the method to be used for selection of the CQI table index.

param table_index FIXed | SEQuence | CONFormance | FOLLow FIXed A fixed mapping table row is used. See also method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsdpa.Cqi.fixed SEQuence A sequence of mapping table rows is used. See also method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.sequence CONFormance A CQI reporting test is to be performed. See also method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsdpa.Cqi.conformance FOLLow The CQI value to be used is proposed by the UE. See also method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.follow

Cloning the Group

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

Subgroups

7.1.12.11.3.1 Conformance

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:CONFormance:MODE
```

class Conformance

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

get_mode() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:CONFormance:MODE
value: bool = driver.configure.cell.hsdpa.cqi.conformance.get_mode()
```

Enables or disables 64QAM modulation in CQI conformance test mode.

return disable_qam_64: OFF | ON

set_mode(*disable_qam_64: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:CONFormance:MODE
driver.configure.cell.hsdpa.cqi.conformance.set_mode(disable_qam_64 = False)
```

Enables or disables 64QAM modulation in CQI conformance test mode.

param disable_qam_64 OFF | ON

7.1.12.11.3.2 RvcSequences

class RvcSequences

RvcSequences 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.cell.hsdpa.cqi.rvcSequences.clone()
```

Subgroups

7.1.12.11.3.3 Qpsk

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:RVCSequences:QPSK:UDEFined
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:RVCSequences:QPSK
```

class Qpsk

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

class UserDefinedStruct

Structure for reading output parameters. Fields:

- Length: int: The first Length entries of the user defined coding sequence are used. Range: 1 to 8
- Sequence: List[int]: Up to 8 values separated by commas. If you specify n values, they overwrite the first n entries of the user-defined sequence. Range: 0 to 7

get_user_defined() → UserDefinedStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CELL:HSDPa:CQI:RVCSequences:QPSK:UDEFined
value: UserDefinedStruct = driver.configure.cell.hsdpa.cqi.rvcSequences.qpsk.
↪get_user_defined()
```

Specifies an RV coding sequence to be used for signals with QPSK modulation if UDEFined is set via method RsCmwWcdmaSig. Configure.Cell.Hsdpa.Cqi.RvcSequences.Qpsk.value.

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

get_value() → RsCmwWcdmaSig.enums.RvcSequence

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RVCSequences:QPSK
value: enums.RvcSequence = driver.configure.cell.hsdpa.cqi.rvcSequences.qpsk.
↪get_value()
```

Specifies an RV coding sequence to be used for signals with QPSK modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.RvcSequences.Qpsk.userDefined.

return sequence: S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

set_user_defined(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Hsdpa_.Cqi_.RvcSequences_.Qpsk.Qpsk.UserDefinedStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CELL:HSDPa:CQI:RVCSequences:QPSK:UDEFined
driver.configure.cell.hsdpa.cqi.rvcSequences.qpsk.set_user_defined(value =␣
↪UserDefinedStruct())
```

Specifies an RV coding sequence to be used for signals with QPSK modulation if UDEFined is set via method RsCmwWcdmaSig. Configure.Cell.Hsdpa.Cqi.RvcSequences.Qpsk.value.

param value see the help for UserDefinedStruct structure arguments.

set_value(*sequence: RsCmwWcdmaSig.enums.RvcSequence*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RVCSequences:QPSK
driver.configure.cell.hsdpa.cqi.rvcSequences.qpsk.set_value(sequence = enums.
↪RvcSequence.S1)
```

Specifies an RV coding sequence to be used for signals with QPSK modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.RvcSequences.Qpsk.userDefined.

param sequence S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

7.1.12.11.3.4 Qam<QuadratureAM>

RepCap Settings

```
# Range: QAM16 .. QAM64
rc = driver.configure.cell.hsdpa.cqi.rvcSequences.qam.repcap_quadratureAM_get()
driver.configure.cell.hsdpa.cqi.rvcSequences.qam.repcap_quadratureAM_set(repcap.
↪QuadratureAM.QAM16)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:RVCSequences:QAM<QuadratureAM>
```

class Qam

Qam commands group definition. 2 total commands, 1 Sub-groups, 1 group commands Repeated Capability: QuadratureAM, default value after init: QuadratureAM.QAM16

get(*quadratureAM=<QuadratureAM.Default: -1>*) → RsCmwWcdmaSig.enums.RvcSequence

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RVCSequences:QAM<nr>
value: enums.RvcSequence = driver.configure.cell.hsdpa.cqi.rvcSequences.qam.
↪get(quadratureAM = repcap.QuadratureAM.Default)
```

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.RvcSequences.Qam.UserDefined.set.

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

return sequence: S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

set(*sequence: RsCmwWcdmaSig.enums.RvcSequence*, *quadratureAM=<QuadratureAM.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RVCSequences:QAM<nr>
driver.configure.cell.hsdpa.cqi.rvcSequences.qam.set(sequence = enums.
↪RvcSequence.S1, quadratureAM = repcap.QuadratureAM.Default)
```

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.Cqi.RvcSequences.Qam.UserDefined.set.

param sequence S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

Cloning the Group

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

Subgroups

7.1.12.11.3.5 UserDefined

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:CQI:RVCSequences:QAM<QuadratureAM>
↪:UDEFined
```

class UserDefined

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

class UserDefinedStruct

Structure for setting input parameters. Fields:

- Length: int: The first Length entries of the user defined coding sequence are used. Range: 1 to 8
- Sequence: List[int]: Up to 8 values separated by commas. If you specify n values, they overwrite the first n entries of the user-defined sequence. Range: 0 to 7

get(*quadratureAM=<QuadratureAM.Default: -1>*) → UserDefinedStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RVCSequences:QAM<nr>
↪:UDEFined
value: UserDefinedStruct = driver.configure.cell.hsdpa.cqi.rvcSequences.qam.
↪userDefined.get(quadratureAM = repcap.QuadratureAM.Default)
```

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation if UDEFined is set via CONFigure:WCDMa:SIGN<i>:CELL:HSDPa:CQI:RVCSequences:QAM<no>.

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

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

set(*structure: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Hsdpa_.Cqi_.RvcSequences_.Qam_.UserDefined.UserDefined.UserDefinedStruct, quadratureAM=<QuadratureAM.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:CQI:RVCSequences:QAM<nr>
↪:UDEFined
driver.configure.cell.hsdpa.cqi.rvcSequences.qam.userDefined.set(value =␣
↪[PROPERTY_STRUCT_NAME](), quadratureAM = repcap.QuadratureAM.Default)
```

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation if UDEFined is set via CONFigure:WCDMa:SIGN<i>:CELL:HSDPa:CQI:RVCSequences:QAM<no>.

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

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

7.1.12.11.4 UserDefined

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UDEFined:HARQ
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UDEFined:IRBuffer
```

class UserDefined

UserDefined commands group definition. 6 total commands, 1 Sub-groups, 2 group commands

get_harq() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UDEFined:HARQ
value: int = driver.configure.cell.hsdpa.userDefined.get_harq()
```

Specifies the number of HARQ processes.

return number: Range: 1 to 8

get_ir_buffer() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UDEFined:IRBuffer
value: int = driver.configure.cell.hsdpa.userDefined.get_ir_buffer()
```

Queries the calculated size (no. of bits) of the virtual IR buffer used in the H-ARQ process.

return buffer_size: Range: 0 bits to 384E+3 bits, Unit: bits

set_harq(*number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UDEFined:HARQ
driver.configure.cell.hsdpa.userDefined.set_harq(number = 1)
```

Specifies the number of HARQ processes.

param number Range: 1 to 8

Cloning the Group

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

Subgroups

7.1.12.11.4.1 RvcSequences

class RvcSequences

RvcSequences 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.cell.hsdpa.userDefined.rvcSequences.clone()
```

Subgroups

7.1.12.11.4.2 Qpsk

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UDEFined:RVCSequences:QPSK:UDEFined
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UDEFined:RVCSequences:QPSK
```

class Qpsk

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

class UserDefinedStruct

Structure for reading output parameters. Fields:

- Length: int: The first Length entries of the user defined coding sequence are used. Range: 1 to 8
- Sequence: List[int]: Up to 8 values separated by commas. If you specify n values, they overwrite the first n entries of the user-defined sequence. Range: 0 to 7

get_user_defined() → UserDefinedStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CELL:HSDPa:UDEFined:RVCSequences:QPSK:UDEFined
value: UserDefinedStruct = driver.configure.cell.hsdpa.userDefined.rvcSequences.
↪qpsk.get_user_defined()
```

Specifies an RV coding sequence to be used for signals with QPSK modulation if UDEFined is set via method RsCmwWcdmaSig. Configure.Cell.Hsdpa.UserDefined.RvcSequences.Qpsk.value.

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

get_value() → RsCmwWcdmaSig.enums.RvcSequence

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CELL:HSDPa:UDEFined:RVCSequences:QPSK
value: enums.RvcSequence = driver.configure.cell.hsdpa.userDefined.rvcSequences.
↪qpsk.get_value()
```

Specifies an RV coding sequence to be used for signals with QPSK modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UserDefined.RvcSequences.Qpsk.userDefined.

return sequence: S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

set_user_defined(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Hsdpa_.UserDefined_.RvcSequences_.Qpsk.Qpsk.UserDefinedSt*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CELL:HSDPa:UDEFined:RVCSequences:QPSK:UDEFined
driver.configure.cell.hsdpa.userDefined.rvcSequences.qpsk.set_user_
↪defined(value = UserDefinedStruct())
```

Specifies an RV coding sequence to be used for signals with QPSK modulation if UDEFined is set via method RsCmwWcdmaSig. Configure.Cell.Hsdpa.UserDefined.RvcSequences.Qpsk.value.

param value see the help for UserDefinedStruct structure arguments.

set_value(*sequence: RsCmwWcdmaSig.enums.RvcSequence*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>
↪:CELL:HSDPa:UDEFined:RVCSequences:QPSK
driver.configure.cell.hsdpa.userDefined.rvcSequences.qpsk.set_value(sequence =␣
↪enums.RvcSequence.S1)
```

(continues on next page)

(continued from previous page)

Specifies an RV coding sequence to be used for signals with QPSK modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UserDefined.RvcSequences.Qpsk.userDefined.

param sequence S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

7.1.12.11.4.3 Qam<QuadratureAM>

RepCap Settings

```
# Range: QAM16 .. QAM64
rc = driver.configure.cell.hsdpa.userDefined.rvcSequences.qam.repcap_quadratureAM_get()
driver.configure.cell.hsdpa.userDefined.rvcSequences.qam.repcap_quadratureAM_set(repcap.
↪QuadratureAM.QAM16)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UDEFined:RVCSequences:QAM<QuadratureAM>
```

class Qam

Qam commands group definition. 2 total commands, 1 Sub-groups, 1 group commands Repeated Capability: QuadratureAM, default value after init: QuadratureAM.QAM16

get(*quadratureAM=<QuadratureAM.Default: -1>*) → RsCmwWcdmaSig.enums.RvcSequence

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UDEFined:RVCSequences:QAM
↪<nr>
value: enums.RvcSequence = driver.configure.cell.hsdpa.userDefined.rvcSequences.
↪qam.get(quadratureAM = repcap.QuadratureAM.Default)
```

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UserDefined.RvcSequences.Qam.UserDefined.set.

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

return sequence: S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

set(*sequence: RsCmwWcdmaSig.enums.RvcSequence*, *quadratureAM=<QuadratureAM.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UDEFined:RVCSequences:QAM
↪<nr>
driver.configure.cell.hsdpa.userDefined.rvcSequences.qam.set(sequence = enums.
↪RvcSequence.S1, quadratureAM = repcap.QuadratureAM.Default)
```

(continues on next page)

(continued from previous page)

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation. If UDEFined is selected, the sequence is defined via method RsCmwWcdmaSig.Configure.Cell.Hsdpa.UserDefined.RvcSequences.Qam.UserDefined.set.

param sequence S1 | S2 | S3 | S4 | S5 | S6 | S7 | UDEFined S1: {0} S2: {6} S3: {0, 2, 5, 6} S4: {6, 2, 1, 5} S5: {0, 0, 0, 0} S6: {6, 6, 6, 6} S7: {6, 0, 4, 5} UDEFined: user-defined sequence

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

Cloning the Group

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

Subgroups

7.1.12.11.4.4 UserDefined

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSDPa:UDEFined:RVCSequences:QAM<QuadratureAM>
↪:UDEFined
```

class UserDefined

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

class UserDefinedStruct

Structure for setting input parameters. Fields:

- Length: int: The first Length entries of the user defined coding sequence are used. Range: 1 to 8
- Sequence: List[int]: Up to 8 values separated by commas. If you specify n values, they overwrite the first n entries of the user-defined sequence. Range: 0 to 7

get(*quadratureAM=<QuadratureAM.Default: -1>*) → UserDefinedStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UDEFined:RVCSequences:QAM
↪<nr>:UDEFined
value: UserDefinedStruct = driver.configure.cell.hsdpa.userDefined.rvcSequences.
↪qam.userDefined.get(quadratureAM = repcap.QuadratureAM.Default)
```

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation if UDEFined is set via CONFigure:WCDMa:SIGN<i>:CELL:HSDPa:UDEFined:RVCSequences:QAM<no>.

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

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

set(*structure: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Hsdpa_.UserDefined_.RvcSequences_.Qam_.UserDefined.UserDefined.UserDefi*
quadratureAM=<QuadratureAM.Default: -1>) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSDPa:UDEFined:RVCSequences:QAM
↪<nr>:UDEFined
driver.configure.cell.hsdpa.userDefined.rvcSequences.qam.userDefined.set(value␣
↪= [PROPERTY_STRUCT_NAME](), quadratureAM = repcap.QuadratureAM.Default)
```

Specifies an RV coding sequence to be used for signals with 16-QAM or 64-QAM modulation if UDEFined is set via CONFigure:WCDMa:SIGN<i>:CELL:HSDPa:UDEFined:RVCSequences:QAM<no>.

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

param quadratureAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Qam')

7.1.12.12 Hsupa

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:TTI
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:HRVersion
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:HBDConition
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:PLPLnonmax
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:MCCode
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:ISGRant
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:MODulation
```

class Hsupa

Hsupa commands group definition. 19 total commands, 6 Sub-groups, 7 group commands

class IsGrantStruct

Structure for reading output parameters. Fields:

- Grant: int or bool: Serving grant value information element Range: 0 to 38 Additional parameters: OFF | ON (disable | enable transmission of the initial serving grant parameters)
- Type_Py: enums.ParameterType: PRIMary | SECondary Primary/secondary grant selector information element

get_hbd_condition() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HBDConition
value: float = driver.configure.cell.hsupa.get_hbd_condition()
```

Specifies the happy bit delay condition value signaled to the UE.

return delay: Only the following values are allowed (in ms) : 2 | 10 | 20 | 50 | 100 | 200 | 500 | 1000 If you enter another value, the nearest allowed value is set instead. Range: 2 ms to 1000 ms, Unit: ms

get_hr_version() → RsCmwWcdmaSig.enums.HrVersion

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HRVersion
value: enums.HrVersion = driver.configure.cell.hsupa.get_hr_version()
```

Specifies the HARQ RV configuration value signaled to the UE.

return version: RV0 | TABLe RV0: use always redundancy version 0 TABLe: determine the redundancy version using a table as specified in 3GPP TS 25.212

get_is_grant() → IsGrantStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:ISGRant
value: IsGrantStruct = driver.configure.cell.hsupa.get_is_grant()
```

Specifies initial serving grant parameters signaled to the UE. If you only want to modify the <Grant>, you can omit the <Type> parameter.

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

get_mccode() → RsCmwWcdmaSig.enums.MaxChanCode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:MCCode
value: enums.MaxChanCode = driver.configure.cell.hsupa.get_mccode()
```

Specifies the maximum channelization codes value signaled to the UE. Depending on several other HSUPA parameters, e.g. the UE category, only a subset of values is allowed.

return code: S64 | S32 | S16 | S8 | S4 | S24 | S22 | S224 S64, S32, S16, S8, S4: one code, SF 64 to SF 4 S24: two codes, SF 4 S22: two codes, SF 2 S224: four codes, two with SF 2 and two with SF 4

get_modulation() → RsCmwWcdmaSig.enums.HsupaModulation

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:MODulation
value: enums.HsupaModulation = driver.configure.cell.hsupa.get_modulation()
```

Selects the E-DCH modulation scheme to be used during HSUPA connection.

return modulation: QPSK | Q16 QPSK, 16-QAM

get_pl_pl_non_max() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:PLPLnonmax
value: float = driver.configure.cell.hsupa.get_pl_pl_non_max()
```

Specifies the 'PLnon-max' value signaled to the UE.

return limit: Range: 0.44 to 1

get_tti() → RsCmwWcdmaSig.enums.TransTimeInterval

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:TTI
value: enums.TransTimeInterval = driver.configure.cell.hsupa.get_tti()
```

Selects the transmission time interval (TTI) for the E-DCH. The value must be compatible with the UE category (2 ms TTI only allowed for category 2, 4 and 6) .

return tti: M2 | M10 M2: 2 ms M10: 10 ms

set_hbd_condition(*delay: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HBDConition
driver.configure.cell.hsupa.set_hbd_condition(delay = 1.0)
```

Specifies the happy bit delay condition value signaled to the UE.

param delay Only the following values are allowed (in ms) : 2 | 10 | 20 | 50 | 100 | 200 | 500 | 1000 If you enter another value, the nearest allowed value is set instead. Range: 2 ms to 1000 ms, Unit: ms

set_hr_version(*version: RsCmwWcdmaSig.enums.HrVersion*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HRVersion
driver.configure.cell.hsupa.set_hr_version(version = enums.HrVersion.RV0)
```

Specifies the HARQ RV configuration value signaled to the UE.

param version RV0 | TABLe RV0: use always redundancy version 0 TABLe: determine the redundancy version using a table as specified in 3GPP TS 25.212

set_is_grant(*value: RsCmwWcdmaSig.Implementations.Configure_.Cell_.Hsupa.Hsupa.IsGrantStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:ISGRant
driver.configure.cell.hsupa.set_is_grant(value = IsGrantStruct())
```

Specifies initial serving grant parameters signaled to the UE. If you only want to modify the <Grant>, you can omit the <Type> parameter.

param value see the help for IsGrantStruct structure arguments.

set_mccode(*code: RsCmwWcdmaSig.enums.MaxChanCode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:MCCode
driver.configure.cell.hsupa.set_mccode(code = enums.MaxChanCode.S16)
```

Specifies the maximum channelization codes value signaled to the UE. Depending on several other HSUPA parameters, e.g. the UE category, only a subset of values is allowed.

param code S64 | S32 | S16 | S8 | S4 | S24 | S22 | S224 S64, S32, S16, S8, S4: one code, SF 64 to SF 4 S24: two codes, SF 4 S22: two codes, SF 2 S224: four codes, two with SF 2 and two with SF 4

set_modulation(*modulation: RsCmwWcdmaSig.enums.HsupaModulation*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:MODulation
driver.configure.cell.hsupa.set_modulation(modulation = enums.HsupaModulation.
↪Q16)
```

Selects the E-DCH modulation scheme to be used during HSUPA connection.

param modulation QPSK | Q16 QPSK, 16-QAM

set_pl_pl_non_max(*limit: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:PLPLnonmax
driver.configure.cell.hsupa.set_pl_pl_non_max(limit = 1.0)
```

Specifies the 'PLnon-max' value signaled to the UE.

param limit Range: 0.44 to 1

set_tti(*tti: RsCmwWcdmaSig.enums.TransTimeInterval*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:TTI
driver.configure.cell.hsupa.set_tti(tti = enums.TransTimeInterval.M10)
```

Selects the transmission time interval (TTI) for the E-DCH. The value must be compatible with the UE category (2 ms TTI only allowed for category 2, 4 and 6) .

param tti M2 | M10 M2: 2 ms M10: 10 ms

Cloning the Group

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

Subgroups

7.1.12.12.1 Pdu

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:PDU:FLEXible
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:PDU
```

class Pdu

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

get_flexible() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:PDU:FLEXible
value: int or bool = driver.configure.cell.hsupa.pdu.get_flexible()
```

Enables and selects the maximum RLC PDU size to be signaled to the UE to configure its flexible UL RLC PDU for dual uplink carrier connections.

return flexible_max: Range: 16 to 12.04E+3 Additional ON/OFF enables/disables flexible PDU size.

get_value() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:PDU
value: int = driver.configure.cell.hsupa.pdu.get_value()
```

Selects the RLC PDU size to be signaled to the UE to configure its constant UL RLC PDU size.

return size: Range: 72 to 5000

set_flexible(*flexible_max: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:PDU:FLEXible
driver.configure.cell.hsupa.pdu.set_flexible(flexible_max = 1)
```

Enables and selects the maximum RLC PDU size to be signaled to the UE to configure its flexible UL RLC PDU for dual uplink carrier connections.

param flexible_max Range: 16 to 12.04E+3 Additional ON/OFF enables/disables flexible PDU size.

set_value(*size: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:PDU
driver.configure.cell.hsupa.pdu.set_value(size = 1)
```

Selects the RLC PDU size to be signaled to the UE to configure its constant UL RLC PDU size.

param size Range: 72 to 5000

7.1.12.12.2 UeCategory

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:UECategory:MANual
```

class UeCategory

UeCategory commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get_manual() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:UECategory:MANual
value: int = driver.configure.cell.hsupa.ueCategory.get_manual()
```

Configures the UE category to be used by the R&S CMW if no reported value is available or usage of the reported value is disabled, see method RsCmwWcdmaSig.Configure.Cell.Hsupa.UeCategory.Reported.set.

return ue_cat_manual: Range: 1 to 9

set_manual(*ue_cat_manual: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:UECategory:MANual
driver.configure.cell.hsupa.ueCategory.set_manual(ue_cat_manual = 1)
```

Configures the UE category to be used by the R&S CMW if no reported value is available or usage of the reported value is disabled, see method RsCmwWcdmaSig.Configure.Cell.Hsupa.UeCategory.Reported.set.

param ue_cat_manual Range: 1 to 9

Cloning the Group

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

Subgroups

7.1.12.12.2.1 Reported

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:UECategory:REPorted
```

class Reported

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

class GetStruct

Response structure. Fields:

- Use_Reported: bool: OFF | ON
- Ue_Cat_Reported: int: UE category reported by the UE (NAV indicates that none has been reported) Range: 1 to 9

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:UECategory:REPorted
value: GetStruct = driver.configure.cell.hsupa.ueCategory.reported.get()
```

Enable or disable usage of the UE category value reported by the UE. When disabled, the UE category must be set manually, see method RsCmwWcdmaSig.Configure.Cell.Hsupa.UeCategory.manual. The manually set value is also used if no reported value is available.

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

set(*use_reported: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:UECategory:REPorted
driver.configure.cell.hsupa.ueCategory.reported.set(use_reported = False)
```

Enable or disable usage of the UE category value reported by the UE. When disabled, the UE category must be set manually, see method RsCmwWcdmaSig.Configure.Cell.Hsupa.UeCategory.manual. The manually set value is also used if no reported value is available.

param use_reported OFF | ON

7.1.12.12.3 Eagch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:EAGCh:TINDex
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:EAGCh:UTTI
```

class Eagch

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

get_tindex() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:EAGCh:TINDex
value: int = driver.configure.cell.hsupa.eagch.get_tindex()
```

Specifies the mapping of the absolute grant value according to 3GPP TS 25.212.

return index: 0: according to table 16B 1: according to table 16B.1, alternative mapping Range: 0 to 1

get_utti() → RsCmwWcdmaSig.enums.UnscheduledTransType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:EAGCh:UTTI
value: enums.UnscheduledTransType = driver.configure.cell.hsupa.eagch.get_utti()
```

Defines the transmission in unscheduled TTIs.

return unscheduled_tti: DUMMy | DTX DUMMy: send absolute grants to dummy UE-IDs DTX: switch E-AGCH off

set_tindex(*index: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:EAGCh:TINDex
driver.configure.cell.hsupa.eagch.set_tindex(index = 1)
```

Specifies the mapping of the absolute grant value according to 3GPP TS 25.212.

param index 0: according to table 16B 1: according to table 16B.1, alternative mapping Range: 0 to 1

set_utti(*unscheduled_tti: RsCmwWcdmaSig.enums.UnscheduledTransType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:EAGCh:UTTI
driver.configure.cell.hsupa.eagch.set_utti(unscheduled_tti = enums.
↪UnscheduledTransType.DTX)
```

Defines the transmission in unscheduled TTIs.

param unscheduled_tti DUMMy | DTX DUMMy: send absolute grants to dummy UE-IDs DTX: switch E-AGCH off

7.1.12.12.4 Horder

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:HORDer:SDCorder
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:HORDer:SUForder
```

class Horder

Horder commands group definition. 3 total commands, 1 Sub-groups, 2 group commands

get_sdc_order() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HORDer:SDCorder
value: bool = driver.configure.cell.hsupa.horder.get_sdc_order()
```

No command help available

return enable: No help available

get_suf_order() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HORDer:SUForder
value: bool = driver.configure.cell.hsupa.horder.get_suf_order()
```

No command help available

return enable: No help available

set_sdc_order(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HORDer:SDCorder
driver.configure.cell.hsupa.horder.set_sdc_order(enable = False)
```

No command help available

param enable No help available

set_suf_order(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HORDer:SUForder
driver.configure.cell.hsupa.horder.set_suf_order(enable = False)
```

No command help available

param enable No help available

Cloning the Group

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

Subgroups

7.1.12.12.4.1 Send

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:HORDer:SEND
```

class Send

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

class GetStruct

Response structure. Fields:

- Frame_Number: int: No parameter help available
- Sfn: int: No parameter help available
- Ack: enums.AckNack: No parameter help available

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HORDer:SEND
value: GetStruct = driver.configure.cell.hsupa.horder.send.get()
```

No command help available

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HORDer:SEND
driver.configure.cell.hsupa.horder.send.set()
```

No command help available

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HORDer:SEND
driver.configure.cell.hsupa.horder.send.set_with_opc()
```

No command help available

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

7.1.12.12.5 Etfci

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:ETFCi:TINDex
```

class Etfci

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

get_tindex() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:ETFCi:TINDex
value: int = driver.configure.cell.hsupa.etfci.get_tindex()
```

Specifies the E-TFCI table index signaled to the UE (use table 0 or table 1 defined in annex B of 3GPP TS 25.321) .

return index: 0 | 1

set_tindex(*index: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:ETFCi:TINDex
driver.configure.cell.hsupa.etfci.set_tindex(index = 1)
```

Specifies the E-TFCI table index signaled to the UE (use table 0 or table 1 defined in annex B of 3GPP TS 25.321) .

param index 0 | 1

7.1.12.12.6 Harq

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:HARQ:POFFset
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HSUPa:HARQ:RETX
```

class Harq

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

get_poffset() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HARQ:POFFset
value: float = driver.configure.cell.hsupa.harq.get_poffset()
```

Specifies the HARQ profile parameter 'E-DCH MAC-d flow power offset' signaled to the UE.

return power_offset: Range: 0 dB to 6 dB, Unit: dB

get_re_tx() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HARQ:RETX
value: int = driver.configure.cell.hsupa.harq.get_re_tx()
```

Specifies the HARQ profile parameter 'E-DCH MAC-d flow maximum number of retransmissions' signaled to the UE.

return number: Range: 0 to 15

set_poffset(*power_offset: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HARQ:POFFset
driver.configure.cell.hsupa.harq.set_poffset(power_offset = 1.0)
```

Specifies the HARQ profile parameter 'E-DCH MAC-d flow power offset' signaled to the UE.

param power_offset Range: 0 dB to 6 dB, Unit: dB

set_re_tx(*number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HSUPa:HARQ:RETX
driver.configure.cell.hsupa.harq.set_re_tx(number = 1)
```

Specifies the HARQ profile parameter 'E-DCH MAC-d flow maximum number of retransmissions' signaled to the UE.

param number Range: 0 to 15

7.1.12.13 Horder

class Horder

Horder 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.cell.horder.clone()
```

Subgroups

7.1.12.13.1 Send

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:HORDer:SEND
```

class Send

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

class GetStruct

Response structure. Fields:

- Frame_Number: int: Information about frame from which the UE has applied the HS-SCCH order
- Sfn: int: Information about subframe from which the UE has applied the HS-SCCH order

- Ack: enums.AckNack: ACK | NACK | DTX ACK: positive acknowledgment NACK: negative acknowledgment DTX: no acknowledgment

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HORDer:SEND
value: GetStruct = driver.configure.cell.horder.send.get()
```

Triggers the HS-SCCH order type 1, according to the preconfiguration of UL/DL and queries the frame number, subframe number and acknowledgment related to the HS-SCCH order execution. For preconfiguration, refer to method RsCmwWcdmaSig. Configure.Cell.Carrier.Horder.downlink etc.

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HORDer:SEND
driver.configure.cell.horder.send.set()
```

Triggers the HS-SCCH order type 1, according to the preconfiguration of UL/DL and queries the frame number, subframe number and acknowledgment related to the HS-SCCH order execution. For preconfiguration, refer to method RsCmwWcdmaSig. Configure.Cell.Carrier.Horder.downlink etc.

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:HORDer:SEND
driver.configure.cell.horder.send.set_with_opc()
```

Triggers the HS-SCCH order type 1, according to the preconfiguration of UL/DL and queries the frame number, subframe number and acknowledgment related to the HS-SCCH order execution. For preconfiguration, refer to method RsCmwWcdmaSig. Configure.Cell.Carrier.Horder.downlink etc.

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

7.1.12.14 Cpc

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:SFORmat
```

class Cpc

Cpc commands group definition. 22 total commands, 6 Sub-groups, 1 group commands

get_sformat() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:SFORmat
value: int = driver.configure.cell.cpc.get_sformat()
```

Configures HS-SCCH less operation to reduce the HS-SCCH overhead and UE battery consumption.

return slot_format: Uplink DPCCH slot format Range: 1 | 4

set_sformat(*slot_format: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:SFORmat
driver.configure.cell.cpc.set_sformat(slot_format = 1)
```

Configures HS-SCCH less operation to reduce the HS-SCCH overhead and UE battery consumption.

param slot_format Uplink DPCCH slot format Range: 1 | 4

Cloning the Group

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

Subgroups

7.1.12.14.1 Dtrx

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:DTRX:DELay
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:DTRX:OFFSet
```

class Dtrx

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

get_delay() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DTRX:DELay
value: int = driver.configure.cell.cpc.dtrx.get_delay()
```

Frame delay the UE waits until enabling a new timing pattern for DRX/DTX operation, see 'Continuous Packet Connectivity (CPC) '.

return enable_delay: Only the following values are allowed (in frames) : 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 If you enter another value, the nearest allowed value is set instead. Range: 0 frames to 128 frames

get_offset() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DTRX:OFFSet
value: int = driver.configure.cell.cpc.dtrx.get_offset()
```

Defines the settings for the discontinuous transmission and reception, see 'Continuous Packet Connectivity (CPC) '.

return offset: Subframe offset to spread the DPCCH transmissions from different UEs Range: 0 Subframe to 159 Subframe

set_delay(*enable_delay: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DTRX:DELay
driver.configure.cell.cpc.dtrx.set_delay(enable_delay = 1)
```

Frame delay the UE waits until enabling a new timing pattern for DRX/DTX operation, see 'Continuous Packet Connectivity (CPC) '.

param enable_delay Only the following values are allowed (in frames) : 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 If you enter another value, the nearest allowed value is set instead. Range: 0 frames to 128 frames

set_offset(*offset: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DTRX:OFFSet
driver.configure.cell.cpc.dtrx.set_offset(offset = 1)
```

Defines the settings for the discontinuous transmission and reception, see 'Continuous Packet Connectivity (CPC) '.

param offset Subframe offset to spread the DPCCH transmissions from different UEs Range: 0 Subframe to 159 Subframe

7.1.12.14.2 Udtx

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:UDTX:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:UDTX:LPLength
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:UDTX:CQITimer
```

class Udtx

Udtx commands group definition. 7 total commands, 1 Sub-groups, 3 group commands

get_cqi_timer() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CQITimer
value: int or bool = driver.configure.cell.cpc.udtx.get_cqi_timer()
```

Number of subframes after an HS-DSCH reception during which the CQI reports have higher priority than the DTX pattern and are transmitted according to the regular CQI pattern, see 'Continuous Packet Connectivity (CPC) '.

return timer: 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | ON | OFF If you enter another value, the nearest allowed value is set instead. Range: 0 Subframe to 512 Subframe, Unit: subframe Additional OFF | ON disables | enables the CQI DTX timer

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:ENABle
value: bool = driver.configure.cell.cpc.udtx.get_enable()
```

Defines the settings for the discontinuous transmission in the uplink, see 'Continuous Packet Connectivity (CPC) '.

return enable: OFF | ON enables/disables UL DTX

get_lp_length() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:LPLength
value: int = driver.configure.cell.cpc.udtx.get_lp_length()
```

Defines the long preamble length that the UE uses during UL DTX cycle 2 to aid synchronization, see 'Continuous Packet Connectivity (CPC) '.

return length: Only the following values are allowed (in slots) : 2 | 4 | 15 If you enter another value, the nearest allowed value is set instead. Range: 2 slots to 15 slots, Unit: slot

set_cqi_timer(*timer: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CQITimer
driver.configure.cell.cpc.udtx.set_cqi_timer(timer = 1)
```

Number of subframes after an HS-DSCH reception during which the CQI reports have higher priority than the DTX pattern and are transmitted according to the regular CQI pattern, see 'Continuous Packet Connectivity (CPC) '.

param timer 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | ON | OFF If you enter another value, the nearest allowed value is set instead. Range: 0 Subframe to 512 Subframe, Unit: subframe Additional OFF | ON disables | enables the CQI DTX timer

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:ENABle
driver.configure.cell.cpc.udtx.set_enable(enable = False)
```

Defines the settings for the discontinuous transmission in the uplink, see 'Continuous Packet Connectivity (CPC) '.

param enable OFF | ON enables/disables UL DTX

set_lp_length(*length: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:LPLength
driver.configure.cell.cpc.udtx.set_lp_length(length = 1)
```

Defines the long preamble length that the UE uses during UL DTX cycle 2 to aid synchronization, see 'Continuous Packet Connectivity (CPC) '.

param length Only the following values are allowed (in slots) : 2 | 4 | 15 If you enter another value, the nearest allowed value is set instead. Range: 2 slots to 15 slots, Unit: slot

Cloning the Group

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

Subgroups

7.1.12.14.2.1 Cycle<Cycle>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.cell.cpc.udtx.cycle.repcap_cycle_get()
driver.configure.cell.cpc.udtx.cycle.repcap_cycle_set(repcap.Cycle.Nr1)
```

class Cycle

Cycle commands group definition. 4 total commands, 4 Sub-groups, 0 group commands Repeated Capability: Cycle, default value after init: Cycle.Nr1

Cloning the Group

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

Subgroups

7.1.12.14.2.2 Apattern

class Apattern

Apattern 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.cell.cpc.udtx.cycle.apattern.clone()
```

Subgroups

7.1.12.14.2.3 Tti<TransTimeInterval>

RepCap Settings

```
# Range: Tti2 .. Tti10
rc = driver.configure.cell.cpc.udtx.cycle.apattern.tti.repcap_transTimeInterval_get()
driver.configure.cell.cpc.udtx.cycle.apattern.tti.repcap_transTimeInterval_set(repcap.
↪TransTimeInterval.Tti2)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:UDTX:CYCLe<Cycle>:APATtern:TTI
↪<TransTimeInterval>
```

class Tti

Tti commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: TransTimeInterval, default value after init: TransTimeInterval.Tti2

get(*cycle=<Cycle.Default: -1>*, *transTimeInterval=<TransTimeInterval.Default: -1>*) → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:APATtern:TTI
↪<ms>
value: int = driver.configure.cell.cpc.udtx.cycle.apattern.tti.get(cycle =␣
↪repcap.Cycle.Default, transTimeInterval = repcap.TransTimeInterval.Default)
```

Defines the UL transmission reduced to DPCCH activity pattern, needed to maintain synchronization and power control loop in the UE DTX cycle, see 'Continuous Packet Connectivity (CPC) '.

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

param transTimeInterval optional repeated capability selector. Default value: Tti2 (settable in the interface 'Tti')

return pattern: Only the following values are allowed for UE DTX cycle 1 (in subframes) : 1 | 5 | 10 | 20 for 10 ms TTI 1 | 4 | 5 | 8 | 10 | 16 | 20 for 2 ms TTI Only the following values are allowed for UE DTX cycle 2 (in subframes) : 5 | 10 | 20 | 40 | 80 | 160 for 10 ms TTI 4 | 5 | 8 | 10 | 16 | 20 | 32 | 40 | 64 | 80 | 128 | 160 for 2 ms TTI If you enter another value, the nearest allowed value is set instead. Range: 1 Subframe to 160 Subframes

set(*pattern: int*, *cycle=<Cycle.Default: -1>*, *transTimeInterval=<TransTimeInterval.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:APATtern:TTI
↪<ms>
driver.configure.cell.cpc.udtx.cycle.apattern.tti.set(pattern = 1, cycle =␣
↪repcap.Cycle.Default, transTimeInterval = repcap.TransTimeInterval.Default)
```

Defines the UL transmission reduced to DPCCH activity pattern, needed to maintain synchronization and power control loop in the UE DTX cycle, see 'Continuous Packet Connectivity (CPC) '.

param pattern Only the following values are allowed for UE DTX cycle 1 (in subframes) : 1 | 5 | 10 | 20 for 10 ms TTI 1 | 4 | 5 | 8 | 10 | 16 | 20 for 2 ms TTI Only the following values are allowed for UE DTX cycle 2 (in subframes) : 5 | 10 | 20 | 40 | 80 | 160 for 10 ms TTI 4 | 5 | 8 | 10 | 16 | 20 | 32 | 40 | 64 | 80 | 128 | 160 for 2 ms TTI If you enter another value, the nearest allowed value is set instead. Range: 1 Subframe to 160 Subframes

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

param transTimeInterval optional repeated capability selector. Default value: Tti2 (settable in the interface 'Tti')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.cell.cpc.udtx.cycle.apattern.tti.clone()
```

7.1.12.14.2.4 Burst

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:UDTX:CYCLe<Cycle>:BURSt
```

class Burst

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

get(*cycle*=<*Cycle.Default: -1*>) → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:BURSt
value: int = driver.configure.cell.cpc.udtx.cycle.burst.get(cycle = repcap.
↪Cycle.Default)
```

Length of DPCCH transmission during UE DTX cycle, see 'Continuous Packet Connectivity (CPC) '.

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

return burst: Only the following values are allowed (in subframes) : 1 | 2 | 5 If you enter another value, the nearest allowed value is set instead. Range: 1 Subframe to 5 Subframe

set(*burst: int*, *cycle*=<*Cycle.Default: -1*>) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:BURSt
driver.configure.cell.cpc.udtx.cycle.burst.set(burst = 1, cycle = repcap.Cycle.
↪Default)
```

Length of DPCCH transmission during UE DTX cycle, see 'Continuous Packet Connectivity (CPC) '.

param burst Only the following values are allowed (in subframes) : 1 | 2 | 5 If you enter another value, the nearest allowed value is set instead. Range: 1 Subframe to 5 Subframe

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

7.1.12.14.2.5 Ithreshold

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:UDTX:CYCLe<Cycle>:ITHReshold
```

class Ithreshold

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

get(*cycle=<Cycle.Default: -1>*) → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:ITHReshold
value: int = driver.configure.cell.cpc.udtx.cycle.ithreshold.get(cycle = repcap.
↪Cycle.Default)
```

Defines when to activate the UE DTX cycle 2 after the last uplink data transmission, see 'Continuous Packet Connectivity (CPC) '.

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

return threshold: Only the following values are allowed (in E-DCH TTI) : 1 | 4 | 8 | 16 | 32 | 64 | 128 | 256 If you enter another value, the nearest allowed value is set instead. Range: 1 E-DCH TTI to 256 E-DCH TTI

set(*threshold: int*, *cycle=<Cycle.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:ITHReshold
driver.configure.cell.cpc.udtx.cycle.ithreshold.set(threshold = 1, cycle =␣
↪repcap.Cycle.Default)
```

Defines when to activate the UE DTX cycle 2 after the last uplink data transmission, see 'Continuous Packet Connectivity (CPC) '.

param threshold Only the following values are allowed (in E-DCH TTI) : 1 | 4 | 8 | 16 | 32 | 64 | 128 | 256 If you enter another value, the nearest allowed value is set instead. Range: 1 E-DCH TTI to 256 E-DCH TTI

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

7.1.12.14.2.6 Dsg

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:UDTX:CYCLe<Cycle>:DSG
```

class Dsg

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

get(*cycle=<Cycle.Default: -1>*) → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:DSG
value: int = driver.configure.cell.cpc.udtx.cycle.dsg.get(cycle = repcap.Cycle.
↪Default)
```

Indicates E-DCH serving grant index to be used in DTX-cycle-2, see 'Continuous Packet Connectivity (CPC) '.

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

return default_sg: 0 to 37: indicates E-DCH serving grant index as defined in 3GPP TS 25.321 38: zero grant Range: 0 to 38

set(*default_sg: int*, *cycle=<Cycle.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:UDTX:CYCLe<nr>:DSG
driver.configure.cell.cpc.udtx.cycle.dsg.set(default_sg = 1, cycle = repcap.
↪Cycle.Default)
```

Indicates E-DCH serving grant index to be used in DTX-cycle-2, see 'Continuous Packet Connectivity (CPC) '.

param default_sg 0 to 37: indicates E-DCH serving grant index as defined in 3GPP TS 25.321 38: zero grant Range: 0 to 38

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

7.1.12.14.3 Ddrx

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:DDRX:ENABle
```

class Ddrx

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

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:ENABle
value: bool = driver.configure.cell.cpc.ddrx.get_enable()
```

Defines the settings for the discontinuous reception in the downlink, see 'Continuous Packet Connectivity (CPC) '.

return enable: OFF | ON enables/disables UE DRX

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:ENABle
driver.configure.cell.cpc.ddrx.set_enable(enable = False)
```

Defines the settings for the discontinuous reception in the downlink, see 'Continuous Packet Connectivity (CPC) '.

param enable OFF | ON enables/disables UE DRX

Cloning the Group

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

Subgroups

7.1.12.14.3.1 Cycle

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:DDRX:CYCLe:APATtern
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:DDRX:CYCLe:ITHReshold
```

class Cycle

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

get_apattern() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:CYCLe:APATtern
value: int = driver.configure.cell.cpc.ddrx.cycle.get_apattern()
```

Reception pattern, to inform UE how often to monitor HS-SCCH, see 'Continuous Packet Connectivity (CPC) '.

return pattern: Only the following values are allowed (in subframes) : 4 | 5 | 8 | 10 | 16 | 20 If you enter another value, the nearest allowed value is set instead. Range: 4 Subframe to 20 Subframe

get_ithreshold() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:CYCLe:ITHReshold
value: int = driver.configure.cell.cpc.ddrx.cycle.get_ithreshold()
```

Number of subframes after downlink activity where UE has to continuously monitor HS-SCCH, see 'Continuous Packet Connectivity (CPC) '.

return threshold: Only the following values are allowed (in subframes) : 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 If you enter another value, the nearest allowed value is set instead. Range: 0 Subframe to 512 Subframe

set_apattern(*pattern: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:CYCLe:APATtern
driver.configure.cell.cpc.ddrx.cycle.set_apattern(pattern = 1)
```

Reception pattern, to inform UE how often to monitor HS-SCCH, see 'Continuous Packet Connectivity (CPC) '.

param pattern Only the following values are allowed (in subframes) : 4 | 5 | 8 | 10 | 16 | 20 If you enter another value, the nearest allowed value is set instead. Range: 4 Subframe to 20 Subframe

set_ithreshold(*threshold: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:CYCLe:ITHReshold
driver.configure.cell.cpc.ddrx.cycle.set_ithreshold(threshold = 1)
```

Number of subframes after downlink activity where UE has to continuously monitor HS-SCCH, see 'Continuous Packet Connectivity (CPC) '.

param threshold Only the following values are allowed (in subframes) : 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 If you enter another value, the nearest allowed value is set instead. Range: 0 Subframe to 512 Subframe

7.1.12.14.3.2 Gmonitoring

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:DDRX:GMONitoring:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:DDRX:GMONitoring:ITHReshold
```

class Gmonitoring

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

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:GMONitoring:ENABle
value: bool = driver.configure.cell.cpc.ddrx.gmonitoring.get_enable()
```

Defines the settings for the discontinuous reception in the downlink, see 'Continuous Packet Connectivity (CPC) '.

return enable: OFF | ON enables/disables UE monitoring of E-AGCH/E-RGCH when they overlap with the start of a UE DRX HS-SCCH reception

get_ithreshold() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:GMONitoring:ITHReshold
value: int = driver.configure.cell.cpc.ddrx.gmonitoring.get_ithreshold()
```

Number of subframes after uplink activity when UE has to monitor E-AGCH/E-RGCH, see 'Continuous Packet Connectivity (CPC) '.

return threshold: Only the following values are allowed (in E-DCH TTIs) : 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 If you enter another value, the nearest allowed value is set instead. Range: 1 E-DCH TTI to 256 E-DCH TTI

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:GMONitoring:ENABle
driver.configure.cell.cpc.ddrx.gmonitoring.set_enable(enable = False)
```

Defines the settings for the discontinuous reception in the downlink, see 'Continuous Packet Connectivity (CPC) '.

param enable OFF | ON enables/disables UE monitoring of E-AGCH/E-RGCH when they overlap with the start of a UE DRX HS-SCCH reception

set_ithreshold(*threshold: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:DDRX:GMONitoring:ITHReshold
driver.configure.cell.cpc.ddrx.gmonitoring.set_ithreshold(threshold = 1)
```

Number of subframes after uplink activity when UE has to monitor E-AGCH/E-RGCH, see 'Continuous Packet Connectivity (CPC) '.

param threshold Only the following values are allowed (in E-DCH TTIs) : 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 If you enter another value, the nearest allowed value is set instead. Range: 1 E-DCH TTI to 256 E-DCH TTI

7.1.12.14.4 Mac

class Mac

Mac 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.cell.cpc.mac.clone()
```

Subgroups

7.1.12.14.4.1 Cycle

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:MAC:CYCLe:ITHReshold
```

class Cycle

Cycle commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get_ithreshold() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:MAC:CYCLe:ITHReshold
value: int or bool = driver.configure.cell.cpc.mac.cycle.get_ithreshold()
```

Restricts the starting points of the uplink transmission on E-DCH for a particular UE. E-DCH inactivity time after which the UE can start E-DCH transmission only at given times, see 'Continuous Packet Connectivity (CPC) '.

return threshold: 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | ON | OFF Values in E-DCH TTIs, additional OFF | ON disables | enables the threshold If you enter another value,

the nearest allowed value is set instead. Range: 1 E-DCH TTI to 512 E-DCH TTI, Unit: E-DCH TTI

set_ithreshold(*threshold: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:MAC:CYCLe:ITHReshold
driver.configure.cell.cpc.mac.cycle.set_ithreshold(threshold = 1)
```

Restricts the starting points of the uplink transmission on E-DCH for a particular UE. E-DCH inactivity time after which the UE can start E-DCH transmission only at given times, see 'Continuous Packet Connectivity (CPC) '.

param threshold 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | ON | OFF Values in E-DCH TTIs, additional OFF | ON disables | enables the threshold If you enter another value, the nearest allowed value is set instead. Range: 1 E-DCH TTI to 512 E-DCH TTI, Unit: E-DCH TTI

Cloning the Group

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

Subgroups

7.1.12.14.4.2 Tti<TransTimeInterval>

RepCap Settings

```
# Range: Tti2 .. Tti10
rc = driver.configure.cell.cpc.mac.cycle.tti.repcap_transTimeInterval_get()
driver.configure.cell.cpc.mac.cycle.tti.repcap_transTimeInterval_set(repcap.
↪TransTimeInterval.Tti2)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:MAC:CYCLe:TTI<TransTimeInterval>
```

class Tti

Tti commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: TransTimeInterval, default value after init: TransTimeInterval.Tti2

get(*transTimeInterval=<TransTimeInterval.Default: -1>*) → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:MAC:CYCLe:TTI<ms>
value: int = driver.configure.cell.cpc.mac.cycle.tti.get(transTimeInterval =␣
↪repcap.TransTimeInterval.Default)
```

Pattern where the start of uplink E-DCH transmission after inactivity is allowed, see 'Continuous Packet Connectivity (CPC) '.

param transTimeInterval optional repeated capability selector. Default value: Tti2 (settable in the interface 'Tti')

return pattern: Only the following values are allowed (in subframes) : 5 | 10 | 20 for 10 ms TTI 1 | 4 | 5 | 8 | 10 | 16 | 20 for 2 ms TTI If you enter another value, the nearest allowed value is set instead. Range: 1 Subframe to 20 Subframe

set(*pattern: int*, *transTimeInterval=<TransTimeInterval.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:MAC:CYCLe:TTI<ms>
driver.configure.cell.cpc.mac.cycle.tti.set(pattern = 1, transTimeInterval =␣
↪repcap.TransTimeInterval.Default)
```

Pattern where the start of uplink E-DCH transmission after inactivity is allowed, see 'Continuous Packet Connectivity (CPC) '.

param pattern Only the following values are allowed (in subframes) : 5 | 10 | 20 for 10 ms TTI 1 | 4 | 5 | 8 | 10 | 16 | 20 for 2 ms TTI If you enter another value, the nearest allowed value is set instead. Range: 1 Subframe to 20 Subframe

param transTimeInterval optional repeated capability selector. Default value: Tti2 (settable in the interface 'Tti')

Cloning the Group

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

7.1.12.14.5 HlOperation

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:HLOPeration:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:HLOPeration:NTBLock
```

class HlOperation

HlOperation commands group definition. 4 total commands, 2 Sub-groups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:ENABle
value: bool = driver.configure.cell.cpc.hlOperation.get_enable()
```

Enables/disables HS-SCCH less operation

return enable: OFF | ON

get_nt_block() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:NTBLock
value: int = driver.configure.cell.cpc.hlOperation.get_nt_block()
```

Selects the number of preconfiguration set (column of the table 'No. of Transport Block Size Indicies') to be used for the initial transmission of HS-SCCH less operation. See also: CONFigure:WCDMa:SIGN<i>:CELL:CPC:HLOPeration:TBLock<index> CONFigure:WCDMa:SIGN<i>:CELL:CPC:HLOPeration:SCSupport<index>

return number: Range: 1 to 4

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:ENABle
driver.configure.cell.cpc.hlOperation.set_enable(enable = False)
```

Enables/disables HS-SCCH less operation

param enable OFF | ON

set_nt_block(*number: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:NTBLock
driver.configure.cell.cpc.hlOperation.set_nt_block(number = 1)
```

Selects the number of preconfiguration set (column of the table 'No. of Transport Block Size Indicies') to be used for the initial transmission of HS-SCCH less operation. See also: CONFigure:WCDMa:SIGN<i>:CELL:CPC:HLOPeration:TBLock<index> CONFigure:WCDMa:SIGN<i>:CELL:CPC:HLOPeration:SCSupport<index>

param number Range: 1 to 4

Cloning the Group

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

Subgroups

7.1.12.14.5.1 Tblock<TransportBlock>

RepCap Settings

```
# Range: TB11 .. TB14
rc = driver.configure.cell.cpc.hlOperation.tblock.repcap_transportBlock_get()
driver.configure.cell.cpc.hlOperation.tblock.repcap_transportBlock_set(repcap.
↪TransportBlock.TB11)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:HLOPeration:TBLock<TransportBlock>
```

class Tblock

Tblock commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: TransportBlock, default value after init: TransportBlock.TBl1

get(*transportBlock=<TransportBlock.Default: -1>*) → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:TBLock<index>
value: List[int] = driver.configure.cell.cpc.hlOperation.tblock.
↪get(transportBlock = repcap.TransportBlock.Default)
```

Predefines the transport block size index for HS-SCCH less operation.

param transportBlock optional repeated capability selector. Default value: TBl1 (settable in the interface 'Tblock')

return index: 1..4 Number of preconfiguration set

set(*index: List[int], transportBlock=<TransportBlock.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:TBLock<index>
driver.configure.cell.cpc.hlOperation.tblock.set(index = [1, 2, 3],␣
↪transportBlock = repcap.TransportBlock.Default)
```

Predefines the transport block size index for HS-SCCH less operation.

param index 1..4 Number of preconfiguration set

param transportBlock optional repeated capability selector. Default value: TBl1 (settable in the interface 'Tblock')

Cloning the Group

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

7.1.12.14.5.2 ScSupport<SecondCode>

RepCap Settings

```
# Range: Sc1 .. Sc4
rc = driver.configure.cell.cpc.hlOperation.scSupport.repcap_secondCode_get()
driver.configure.cell.cpc.hlOperation.scSupport.repcap_secondCode_set(repcap.SecondCode.
↪Sc1)
```

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:HLOPeration:SCSupport<SecondCode>
```

class ScSupport

ScSupport commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: SecondCode, default value after init: SecondCode.Sc1

get(*secondCode=<SecondCode.Default: -1>*) → List[bool]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:SCSupport
↪<index>
value: List[bool] = driver.configure.cell.cpc.hlOperation.scSupport.
↪get(secondCode = repcap.SecondCode.Default)
```

Predefines the support of HS-PDSCH second code for HS-SCCH less operation.

param secondCode optional repeated capability selector. Default value: Sc1 (settable in the interface 'ScSupport')

return enable: OFF | ON

set(*enable: List[bool], secondCode=<SecondCode.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HLOPeration:SCSupport
↪<index>
driver.configure.cell.cpc.hlOperation.scSupport.set(enable = [True, False,␣
↪True], secondCode = repcap.SecondCode.Default)
```

Predefines the support of HS-PDSCH second code for HS-SCCH less operation.

param enable OFF | ON

param secondCode optional repeated capability selector. Default value: Sc1 (settable in the interface 'ScSupport')

Cloning the Group

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

7.1.12.14.6 Horder

class Horder

Horder 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.cell.cpc.horder.clone()
```

Subgroups

7.1.12.14.6.1 Send

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CELL:CPC:HORDer:SEND
```

class Send

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

class GetStruct

Response structure. Fields:

- Frame_Number: int: Information about frame from which the UE has applied the HS-SCCH order
- Sfn: int: Information about subframe from which the UE has applied the HS-SCCH order
- Ack: enums.AckNack: ACK | NACK | DTX ACK: positive acknowledgment NACK: negative acknowledgment DTX: no acknowledgment

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HORDer:SEND
value: GetStruct = driver.configure.cell.cpc.horder.send.get()
```

Tells the UE to enable/disable discontinuous downlink reception and/or discontinuous uplink DPCCH transmission and queries the frame number, subframe number and acknowledgment related to the HS-SCCH order type 0 execution. See also 'Continuous Packet Connectivity (CPC) '.

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

set() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HORDer:SEND
driver.configure.cell.cpc.horder.send.set()
```

Tells the UE to enable/disable discontinuous downlink reception and/or discontinuous uplink DPCCH transmission and queries the frame number, subframe number and acknowledgment related to the HS-SCCH order type 0 execution. See also 'Continuous Packet Connectivity (CPC) '.

set_with_opc() → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CELL:CPC:HORDer:SEND
driver.configure.cell.cpc.horder.send.set_with_opc()
```

Tells the UE to enable/disable discontinuous downlink reception and/or discontinuous uplink DPCCH transmission and queries the frame number, subframe number and acknowledgment related to the HS-SCCH order type 0 execution. See also 'Continuous Packet Connectivity (CPC) '.

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

7.1.13 Ncell<Cell>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.configure.ncell.repcap_cell_get()
driver.configure.ncell.repcap_cell_set(repcap.Cell.Nr1)
```

class Ncell

Ncell commands group definition. 4 total commands, 3 Sub-groups, 0 group commands Repeated Capability: Cell, default value after init: Cell.Nr1

Cloning the Group

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

Subgroups

7.1.13.1 Lte

class Lte

Lte 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.ncell.lte.clone()
```

Subgroups

7.1.13.1.1 Cell

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:NCELl:LTE:CELL<Cell>
```

class Cell

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

class CellStruct

Structure for setting input parameters. Contains optional setting 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 | OB45 | OB46 | OB65 | OB66 | OB67 | OB252 | OB255 Operating band 1 to 46, 65 to 67, 252, 255
- Channel: int: Downlink channel number Range: depends on operating band, see tables below
- Measurement: bool: Optional setting parameter. OFF | ON Enables or disables the UE measurement

get(*cell=<Cell.Default: -1>*) → CellStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:LTE:CELL<n>
value: CellStruct = driver.configure.ncell.lte.cell.get(cell = repcap.Cell.
↪Default)
```

Configures an entry of the neighbor cell list for LTE.

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

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

set(*structure: RsCmwWcdmaSig.Implementations.Configure_.Ncell_.Lte_.Cell.Cell.CellStruct*, *cell=<Cell.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:LTE:CELL<n>
driver.configure.ncell.lte.cell.set(value = [PROPERTY_STRUCT_NAME](), cell =␣
↪repcap.Cell.Default)
```

Configures an entry of the neighbor cell list for LTE.

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

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

7.1.13.1.2 Thresholds

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:NCELl:LTE:THResholds:HIGH
```

class Thresholds

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

get_high() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:LTE:THResholds:HIGH
value: int = driver.configure.ncell.lte.thresholds.get_high()
```

Configures the reselection threshold value 'threshXhigh' for LTE neighbor cells.

return high: Range: 0 to 31

set_high(*high: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:LTE:THResholds:HIGH
driver.configure.ncell.lte.thresholds.set_high(high = 1)
```

Configures the reselection threshold value 'threshXhigh' for LTE neighbor cells.

param high Range: 0 to 31

7.1.13.2 Gsm

class Gsm

Gsm 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.ncell.gsm.clone()
```

Subgroups

7.1.13.2.1 Cell

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:NCELl:GSM:CELL<Cell>
```

class Cell

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

class CellStruct

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

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.GsmBand: G04 | G085 | G09 | G18 | G19 GSM 400, GSM 850, GSM 900, GSM 1800, GSM 1900
- Channel: int: Channel number used for the broadcast control channel (BCCH) Range: 0 to 1023, depending on GSM band, see table below
- Measurement: bool: Optional setting parameter. OFF | ON Enables or disables the UE measurement
- Bsic: int: Optional setting parameter. Base station identity code Range: 0 to 63

get(*cell=<Cell.Default: -1>*) → CellStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:GSM:CELL<n>
value: CellStruct = driver.configure.ncell.gsm.cell.get(cell = repcap.Cell.
↪Default)
```

Configures an entry of the neighbor cell list for GSM.

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

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

set(*structure: RsCmwWcdmaSig.Implementations.Configure_.Ncell_.Gsm_.Cell.Cell.CellStruct*, *cell=<Cell.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:GSM:CELL<n>
driver.configure.ncell.gsm.cell.set(value = [PROPERTY_STRUCT_NAME](), cell =␣
↪repcap.Cell.Default)
```

Configures an entry of the neighbor cell list for GSM.

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

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

7.1.13.3 Wcdma

class Wcdma

Wcdma 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.ncell.wcdma.clone()
```

Subgroups

7.1.13.3.1 Cell

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:NCELl:WCDMa:CELL<Cell>
```

class Cell

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

class CellStruct

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

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.OperationBand: OB1 | … | OB14 | OB19 | … | OB22 | OB25 | OB26 | OBS1 | … | OBS3 | OBL1 OB1, …, OB14: operating band I to XIV OB19, …, OB22: operating band XIX to XXII OB25, OB26: operating band XXV, XXVI OB32: operating band XXXII (restricted to dual band scenarios) OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L
- Channel: int: Downlink channel number Range: depends on operating band
- Scrambling_Code: str: Primary scrambling code Range: #H0 to #H1FF

- Measurement: bool: Optional setting parameter. OFF | ON Enables or disables the UE measurement

get(*cell=<Cell.Default: -1>*) → CellStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:WCDMa:CELL<n>
value: CellStruct = driver.configure.ncell.wcdma.cell.get(cell = repcap.Cell.
↪Default)
```

Configures an entry of the neighbor cell list for WCDMA. For channel number ranges depending on operating bands see 'Operating Bands'.

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

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

set(*structure: RsCmwWcdmaSig.Implementations.Configure_.Ncell_.Wcdma_.Cell.Cell.CellStruct*, *cell=<Cell.Default: -1>*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:NCELl:WCDMa:CELL<n>
driver.configure.ncell.wcdma.cell.set(value = [PROPERTY_STRUCT_NAME](), cell =␣
↪repcap.Cell.Default)
```

Configures an entry of the neighbor cell list for WCDMA. For channel number ranges depending on operating bands see 'Operating Bands'.

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

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

7.1.14 Ber

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:BER:PNResync
CONFigure:WCDMa:SIGNaling<Instance>:BER:LIMit
CONFigure:WCDMa:SIGNaling<Instance>:BER:TBLocks
CONFigure:WCDMa:SIGNaling<Instance>:BER:SCONdition
CONFigure:WCDMa:SIGNaling<Instance>:BER:REPetition
CONFigure:WCDMa:SIGNaling<Instance>:BER:TOUT
```

class Ber

Ber commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

class LimitStruct

Structure for reading output parameters. Fields:

- Ber: float or bool: Range: 0 % to 100 %, Unit: % Additional OFF | ON disables | enables the limit using the previous/default level
- Bler: float or bool: Range: 0 % to 100 %, Unit: % Additional OFF | ON disables | enables the limit using the previous/default level
- Dbler: float or bool: Range: 0 % to 100 %, Unit: % Additional OFF | ON disables | enables the limit using the previous/default level

- Lost_Trans_Blocks: int or bool: Range: 1 to 50000 Additional OFF | ON disables | enables the limit using the previous/default level
- Ult_Fci_Faults: float or bool: Range: 0 % to 100 %, Unit: % Additional OFF | ON disables | enables the limit using the previous/default level
- Fdr: float or bool: Range: 0 % to 100 %, Unit: % Additional OFF | ON disables | enables the limit using the previous/default level
- Pn_Discontinuity: int or bool: Range: 1 to 50000 Additional OFF | ON disables | enables the limit using the previous/default level

get_limit() → LimitStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:LIMit
value: LimitStruct = driver.configure.ber.get_limit()
```

Specifies upper limits for the results of the 'BER' measurement.

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

get_pn_resync() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:PNResync
value: bool = driver.configure.ber.get_pn_resync()
```

Activates or deactivates a correction (reordering) mechanism for transports blocks looped back in wrong order.

return enable: OFF | ON ON: correction mechanism active, BER measurement result based on corrected block sequence, number of corrected blocks available as result PN discontinuity OFF: correction mechanism inactive, no PN discontinuity result

get_repetition() → RsCmwWcdmaSig.enums.Repeat

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:REPetition
value: enums.Repeat = driver.configure.ber.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Ber.tblocks to determine the number of transport blocks per single shot.

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

get_scondition() → RsCmwWcdmaSig.enums.StopCondition

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:SCONdition
value: enums.StopCondition = driver.configure.ber.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 | SLFail NONE: Continue measurement irrespective of the limit check SLFail: Stop measurement on limit failure

get_tblocks() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:TBLocks
value: int = driver.configure.ber.get_tblocks()
```

Defines the number of transport blocks to be measured per measurement cycle (statistics cycle) .

return transport_blocks: Range: 1 to 50E+3

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:TOUT
value: float = driver.configure.ber.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_limit(*value: RsCmwWcdmaSig.Implementations.Configure_.Ber.Ber.LimitStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:LIMit
driver.configure.ber.set_limit(value = LimitStruct())
```

Specifies upper limits for the results of the 'BER' measurement.

param value see the help for LimitStruct structure arguments.

set_pn_resync(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:PNResync
driver.configure.ber.set_pn_resync(enable = False)
```

Activates or deactivates a correction (reordering) mechanism for transports blocks looped back in wrong order.

param enable OFF | ON ON: correction mechanism active, BER measurement result based on corrected block sequence, number of corrected blocks available as result PN discontinuity OFF: correction mechanism inactive, no PN discontinuity result

set_repetition(*repetition: RsCmwWcdmaSig.enums.Repeat*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:REPetition
driver.configure.ber.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Ber.tblocks to determine the number of transport blocks per single shot.

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

set_scondition(*stop_condition: RsCmwWcdmaSig.enums.StopCondition*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:SCONdition
driver.configure.ber.set_scondition(stop_condition = enums.StopCondition.NONE)
```

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

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

set_tblocks(*transport_blocks: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:TBLocks
driver.configure.ber.set_tblocks(transport_blocks = 1)
```

Defines the number of transport blocks to be measured per measurement cycle (statistics cycle) .

param transport_blocks Range: 1 to 50E+3

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:BER:TOUT
driver.configure.ber.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

7.1.15 Throughput

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:THRoughput:TOUT
CONFigure:WCDMa:SIGNaling<Instance>:THRoughput:UPDate
CONFigure:WCDMa:SIGNaling<Instance>:THRoughput:WINDow
CONFigure:WCDMa:SIGNaling<Instance>:THRoughput:REPetition
```

class Throughput

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

get_repetition() → RsCmwWcdmaSig.enums.Repeat

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:REPetition
value: enums.Repeat = driver.configure.throughput.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Throughput.window to configure the duration of a single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:TOUT
value: float = driver.configure.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

get_update() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:UPDate
value: float = driver.configure.throughput.get_update()
```

Configures the time interval used to derive a single throughput result.

return interval: Range: 0.24 s / 120 subframes to 2.4 s / 1200 subframes, Unit: subframe

get_window() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:WINDow
value: float = driver.configure.throughput.get_window()
```

Specifies the duration of a single-shot measurement, i.e. the time interval covered by a throughput result trace. The value is internally rounded up to the next integer multiple of the time interval used to calculate a single result (see method RsCmwWcdmaSig.Configure.Throughput.update) .

return size: Range: 9.6 s / 48000 subframes to 240 s / 120000 subframes, Unit: subframe

set_repetition(*repetition: RsCmwWcdmaSig.enums.Repeat*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:REPetition
driver.configure.throughput.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Throughput.window to configure the duration of a single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:TOUT
driver.configure.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

set_update(*interval: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:UPDate
driver.configure.throughput.set_update(interval = 1.0)
```

Configures the time interval used to derive a single throughput result.

param interval Range: 0.24 s / 120 subframes to 2.4 s / 1200 subframes, Unit: subframe

set_window(*size: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:THRoughput:WINDow
driver.configure.throughput.set_window(size = 1.0)
```

Specifies the duration of a single-shot measurement, i.e. the time interval covered by a throughput result trace. The value is internally rounded up to the next integer multiple of the time interval used to calculate a single result (see method RsCmwWcdmaSig.Configure.Throughput.update) .

param size Range: 9.6 s / 48000 subframes to 240 s / 120000 subframes, Unit: subframe

7.1.16 Hack

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:HACK:TOUT
CONFigure:WCDMa:SIGNaling<Instance>:HACK:HARQ
CONFigure:WCDMa:SIGNaling<Instance>:HACK:REPetition
CONFigure:WCDMa:SIGNaling<Instance>:HACK:MSFRames
```

class Hack

Hack commands group definition. 5 total commands, 1 Sub-groups, 4 group commands

get_harq() → RsCmwWcdmaSig.enums.MonitoredHarq

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:HARQ
value: enums.MonitoredHarq = driver.configure.hack.get_harq()
```

Selects either a single H-ARQ process (numbered 0 to 7) to be monitored or specifies that all processes are to be monitored.

return monitored_harq: ALL | H0 | H1 | H2 | H3 | H4 | H5 | H6 | H7

get_ms_frames() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:MSFRames
value: int = driver.configure.hack.get_ms_frames()
```

Defines the number of HSDPA subframes to be measured per measurement cycle (statistics cycle) .

return meas_sub_frames: Range: 100 to 1E+6

get_repetition() → RsCmwWcdmaSig.enums.Repeat

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:REPetition
value: enums.Repeat = driver.configure.hack.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Hack.msFrames to determine the number of HSDPA subframes to be measured per single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:TOUT
value: float = driver.configure.hack.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_harq(*monitored_harq: RsCmwWcdmaSig.enums.MonitoredHarq*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:HARQ
driver.configure.hack.set_harq(monitored_harq = enums.MonitoredHarq.ALL)
```

Selects either a single H-ARQ process (numbered 0 to 7) to be monitored or specifies that all processes are to be monitored.

param monitored_harq ALL | H0 | H1 | H2 | H3 | H4 | H5 | H6 | H7

set_ms_frames(*meas_sub_frames: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:MSFRames
driver.configure.hack.set_ms_frames(meas_sub_frames = 1)
```

Defines the number of HSDPA subframes to be measured per measurement cycle (statistics cycle) .

param meas_sub_frames Range: 100 to 1E+6

set_repetition(*repetition: RsCmwWcdmaSig.enums.Repeat*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:REPetition
driver.configure.hack.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Hack.msFrames to determine the number of HSDPA subframes to be measured per single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:TOUT
driver.configure.hack.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

Cloning the Group

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

Subgroups

7.1.16.1 Smode

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:HACK:SMODe:AVERage
```

class Smode

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

get_average() → RsCmwWcdmaSig.enums.AveragingMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:SMODe:AVERage
value: enums.AveragingMode = driver.configure.hack.smode.get_average()
```

Specifies calculation algorithm for average statistics. Only remote command is provided. No corresponding manual setting is existing in the GUI. The setting is especially useful if 'Repetition' = 'Continuous'.

return mode: WINDow | CONTinuous WINDow: average results calculated per statistic cycle ('Measure Subframes') CONTinuous: average results calculated since the beginning of the measurement

set_average(*mode: RsCmwWcdmaSig.enums.AveragingMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HACK:SMODe:AVERage
driver.configure.hack.smode.set_average(mode = enums.AveragingMode.CONTinuous)
```

Specifies calculation algorithm for average statistics. Only remote command is provided. No corresponding manual setting is existing in the GUI. The setting is especially useful if 'Repetition' = 'Continuous'.

param mode WINDow | CONTinuous WINDow: average results calculated per statistic cycle ('Measure Subframes') CONTinuous: average results calculated since the beginning of the measurement

7.1.17 Hcqi

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:TOUT
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:TCASe
```

class Hcqi

Hcqi commands group definition. 9 total commands, 3 Sub-groups, 2 group commands

get_tcase() → RsCmwWcdmaSig.enums.TestCase

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:TCASe
value: enums.TestCase = driver.configure.hcqi.get_tcase()
```

Selects either AWGN or fading test case.

return test_case: AWGN | FADing

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:TOUT
value: float = driver.configure.hcqi.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_tcase(*test_case: RsCmwWcdmaSig.enums.TestCase*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:TCASe
driver.configure.hcqi.set_tcase(test_case = enums.TestCase.AWGN)
```

Selects either AWGN or fading test case.

param test_case AWGN | FADing

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:TOUT
driver.configure.hcqi.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

Cloning the Group

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

Subgroups

7.1.17.1 Cqi

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:CQI:MSFRames
```

class Cqi

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

get_ms_frames() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:CQI:MSFRames
value: int = driver.configure.hcqi.cqi.get_ms_frames()
```

Defines the number of HSDPA subframes for the first measurement stage to be measured per measurement cycle (statistics cycle) .

return meas_sub_frames: Range: 100 to 1E+6

set_ms_frames(*meas_sub_frames: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:CQI:MSFRames
driver.configure.hcqi.cqi.set_ms_frames(meas_sub_frames = 1)
```

Defines the number of HSDPA subframes for the first measurement stage to be measured per measurement cycle (statistics cycle) .

param meas_sub_frames Range: 100 to 1E+6

7.1.17.2 Bler

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:BLER:MSFRames
```

class Bler

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

get_ms_frames() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:BLER:MSFRames
value: int = driver.configure.hcqi.bler.get_ms_frames()
```

Defines the number of HSDPA subframes for the second measurement stage to be measured per measurement cycle (statistics cycle) .

return meas_sub_frames: Range: 100 to 1E+6

set_ms_frames(*meas_sub_frames: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:BLER:MSFRames
driver.configure.hcqi.bler.set_ms_frames(meas_sub_frames = 1)
```

Defines the number of HSDPA subframes for the second measurement stage to be measured per measurement cycle (statistics cycle) .

param meas_sub_frames Range: 100 to 1E+6

7.1.17.3 Limit

class Limit

Limit commands group definition. 5 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.17.3.1 Awgn

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:LIMit:AWGN:BLER
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:LIMit:AWGN:DTX
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:LIMit:AWGN
```

class Awgn

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

class BlerStruct

Structure for reading output parameters. Fields:

- Median_M_1: float: Upper limit for the values acquired at median CQI - 1. This limit applies if BLER at median CQI is above the limit Median0. Range: 0 % to 100 %, Unit: %
- Median_0: float: Limit for the values acquired at median CQI Range: 0 % to 100 %, Unit: %
- Median_P_2: float: Lower limit for the values acquired at median CQI + 2. This limit applies if BLER at median CQI is below the limit Median0. Range: 0 % to 100 %, Unit: %

class DtxStruct

Structure for reading output parameters. Fields:

- Median_M_1: float or bool: Limit for the values acquired at median CQI - 1 Range: 0 % to 100 % Additional OFF | ON disables | enables the limit check

- Median_0: float or bool: Limit for the values acquired at median CQI Range: 0 % to 100 % Additional OFF | ON disables | enables the limit check
- Median_P_2: float or bool: Limit for the values acquired at median CQI + 2 Range: 0 % to 100 % Additional OFF | ON disables | enables the limit check

get_bler() → BlerStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:AWGN:BLER
value: BlerStruct = driver.configure.hcqi.limit.awgn.get_bler()
```

Defines BLER limit for AWGN test case.

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

get_dtx() → DtxStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:AWGN:DTX
value: DtxStruct = driver.configure.hcqi.limit.awgn.get_dtx()
```

Defines the maximum percentage of HSDPA subframes that the UE answers with DTX during AWGN test case.

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

get_value() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:AWGN
value: float = driver.configure.hcqi.limit.awgn.get_value()
```

Specifies the minimum percentage of measured CQI values, that fall in the range (median CQI – 2) median CQI (median CQI + 2) .

return cqiin_range: Lower limit for the first stage of AWGN test case Range: 0 % to 100 %, Unit: %

set_bler(*value: RsCmwWcdmaSig.Implementations.Configure_.Hcqi_.Limit_.Awgn.Awgn.BlerStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:AWGN:BLER
driver.configure.hcqi.limit.awgn.set_bler(value = BlerStruct())
```

Defines BLER limit for AWGN test case.

param value see the help for BlerStruct structure arguments.

set_dtx(*value: RsCmwWcdmaSig.Implementations.Configure_.Hcqi_.Limit_.Awgn.Awgn.DtxStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:AWGN:DTX
driver.configure.hcqi.limit.awgn.set_dtx(value = DtxStruct())
```

Defines the maximum percentage of HSDPA subframes that the UE answers with DTX during AWGN test case.

param value see the help for DtxStruct structure arguments.

set_value(*cqiin_range: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:AWGN
driver.configure.hcqi.limit.awgn.set_value(cqiin_range = 1.0)
```

Specifies the minimum percentage of measured CQI values, that fall in the range (median CQI – 2) median CQI (median CQI + 2) .

param cqiin_range Lower limit for the first stage of AWGN test case Range: 0 % to 100 %, Unit: %

7.1.17.3.2 Fading

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:LIMit:FADing:BLER
CONFigure:WCDMa:SIGNaling<Instance>:HCQI:LIMit:FADing:DTX
```

class Fading

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

class BlerStruct

Structure for reading output parameters. Fields:

- Median_0: float: Limit for the values acquired at median CQI Range: 0 % to 100 %, Unit: %
- Median_P_3: float: Limit for the values acquired at median CQI + 3 Range: 0 % to 100 %, Unit: %

class DtxStruct

Structure for reading output parameters. Fields:

- Median_0: float or bool: Limit for the values acquired at median CQI Range: 0 % to 100 % Additional parameters: OFF | ON (disables | enables the limit check)
- Median_P_3: float or bool: Limit for the values acquired at median CQI + 3 Range: 0 % to 100 % Additional parameters: OFF | ON (disables | enables the limit check)

get_bler() → BlerStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:FADing:BLER
value: BlerStruct = driver.configure.hcqi.limit.fading.get_bler()
```

Defines upper BLER limit for fading test case.

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

get_dtx() → DtxStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:FADing:DTX
value: DtxStruct = driver.configure.hcqi.limit.fading.get_dtx()
```

Defines the maximum percentage of HSDPA subframes that the UE answers with DTX during fading test case.

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

set_bler(*value: RsCmwWcdmaSig.Implementations.Configure_.Hcqi_.Limit_.Fading.Fading.BlerStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:FADing:BLER
driver.configure.hcqi.limit.fading.set_bler(value = BlerStruct())
```

Defines upper BLER limit for fading test case.

param value see the help for BlerStruct structure arguments.

set_dtx(*value: RsCmwWcdmaSig.Implementations.Configure_.Hcqi_.Limit_.Fading.Fading.DtxStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:HCQI:LIMit:FADing:DTX
driver.configure.hcqi.limit.fading.set_dtx(value = DtxStruct())
```

Defines the maximum percentage of HSDPA subframes that the UE answers with DTX during fading test case.

param value see the help for DtxStruct structure arguments.

7.1.18 UplinkLogging

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:ULLogging:TOUT
CONFigure:WCDMa:SIGNaling<Instance>:ULLogging:SCCYcle
CONFigure:WCDMa:SIGNaling<Instance>:ULLogging:SSFN
CONFigure:WCDMa:SIGNaling<Instance>:ULLogging:MSFRames
CONFigure:WCDMa:SIGNaling<Instance>:ULLogging:REPetition
```

class UplinkLogging

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

get_ms_frames() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:MSFRames
value: int = driver.configure.uplinkLogging.get_ms_frames()
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) .

return meas_sub_frames: Volume of measured consecutive UL HS-DPCCH/E-DPCCH/DPCCH subframes Range: 15 to 10E+3

get_repetition() → RsCmwWcdmaSig.enums.Repeat

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:REPetition
value: enums.Repeat = driver.configure.uplinkLogging.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames to define the number of subframes to be measured per single shot.

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

get_sccycle() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:SCCYcle
value: bool = driver.configure.uplinkLogging.get_sccycle()
```

Enables in the UL logging RX measurement to be started two subframes before a CPC cycle one.

return enable: OFF | ON

get_ssfn() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:SSFN
value: int or bool = driver.configure.uplinkLogging.get_ssfn()
```

Specifies the first system frame number for which the UL HS-DPCCH/E-DPCCH/DPCCH information is displayed. System frame number corresponds to the subframe number of the UL HS-DPCCH/E-DPCCH/DPCCH.

return sfn: First system frame number set to modulo 4095 Range: 0 to 4095 Additional ON / OFF enables or disables the use of SFN.

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:TOUT
value: float = driver.configure.uplinkLogging.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_ms_frames(*meas_sub_frames: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:MSFRames
driver.configure.uplinkLogging.set_ms_frames(meas_sub_frames = 1)
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) .

param meas_sub_frames Volume of measured consecutive UL HS-DPCCH/E-DPCCH/DPCCH subframes Range: 15 to 10E+3

set_repetition(*repetition: RsCmwWcdmaSig.enums.Repeat*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:REPetition
driver.configure.uplinkLogging.set_repetition(repetition = enums.Repeat.
↪CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames to define the number of subframes to be measured per single shot.

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

set_sccycle(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:SCCYcle
driver.configure.uplinkLogging.set_sccycle(enable = False)
```

Enables in the UL logging RX measurement to be started two subframes before a CPC cycle one.

param enable OFF | ON

set_ssfn(*sfn: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:SSFN
driver.configure.uplinkLogging.set_ssfn(sfn = 1)
```

Specifies the first system frame number for which the UL HS-DPCCH/E-DPCCH/DPCCH information is displayed. System frame number corresponds to the subframe number of the UL HS-DPCCH/E-DPCCH/DPCCH.

param sfn First system frame number set to modulo 4095 Range: 0 to 4095 Additional ON / OFF enables or disables the use of SFN.

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ULLogging:TOUT
driver.configure.uplinkLogging.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

7.1.19 Eagch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:TOUT
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:REPetition
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:MFRames
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:MTYPe
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:LIMit
```

class Eagch

Eagch commands group definition. 8 total commands, 1 Sub-groups, 5 group commands

get_limit() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:LIMit
value: float = driver.configure.eagch.get_limit()
```

Upper limit for the ratio of missed detections to the detected E-TFCI events.

return probability: Range: 0 % to 100 %

get_mframes() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:MFRames
value: int = driver.configure.eagch.get_mframes()
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) . Ideally, one E-TFCI value is detected per TTI.

return meas_frames: Range: 1 to 1E+6

get_mtype() → RsCmwWcdmaSig.enums.MeasType

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:MTYPe
value: enums.MeasType = driver.configure.eagch.get_mtype()
```

Specifies the type of measurement.

return meas_type: GENeral | MISSed General histogram or missed detection

get_repetition() → RsCmwWcdmaSig.enums.Repeat

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:REPetition
value: enums.Repeat = driver.configure.eagch.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Eagch.mframes to define the number of subframes to be measured per single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:TOUT
value: float = driver.configure.eagch.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: No help available

set_limit(*probability: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:LIMit
driver.configure.eagch.set_limit(probability = 1.0)
```

Upper limit for the ratio of missed detections to the detected E-TFCI events.

param probability Range: 0 % to 100 %

set_mframes(*meas_frames: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:MFRames
driver.configure.eagch.set_mframes(meas_frames = 1)
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) . Ideally, one E-TFCI value is detected per TTI.

param meas_frames Range: 1 to 1E+6

set_mtype(*meas_type: RsCmwWcdmaSig.enums.MeasType*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:MTYPe
driver.configure.eagch.set_mtype(meas_type = enums.MeasType.GENeral)
```

Specifies the type of measurement.

param meas_type GENeral | MISSed General histogram or missed detection

set_repetition(*repetition: RsCmwWcdmaSig.enums.Repeat*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:REPetition
driver.configure.eagch.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Eagch.mframes to define the number of subframes to be measured per single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:TOUT
driver.configure.eagch.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 No help available

Cloning the Group

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

Subgroups

7.1.19.1 Etfci

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:ETFCi:MODE
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:ETFCi:MANual
CONFigure:WCDMa:SIGNaling<Instance>:EAGCh:ETFCi:AUTO
```

class Etfci

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

get_auto() → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:ETFCi:AUTO
value: List[int] = driver.configure.eagch.etfci.get_auto()
```

Queries the n E-TFCI values calculated according to the absolute grant (AG) configuration. The number of values n equals AG pattern length, see method RsCmwWcdmaSig.Configure.Cell.Carrier.Hsupa.Eagch.Pattern.length

return etfci: Range: 0 to 127

get_manual() → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:ETFCi:MANual
value: List[int] = driver.configure.eagch.etfci.get_manual()
```

Specifies up to eight E-TFCI values used for E-AGCH measurement in manual mode.

return etfci: Range: 0 to 127

get_mode() → RsCmwWcdmaSig.enums.AutoManualMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:ETFCi:MODE
value: enums.AutoManualMode = driver.configure.eagch.etfci.get_mode()
```

Specifies the mode of expected E-TFCI selection.

return mode: AUTO | MANual Automatic according to AG pattern or manual

set_manual(*etfci: List[int]*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:ETFCi:MANual
driver.configure.eagch.etfci.set_manual(etfci = [1, 2, 3])
```

Specifies up to eight E-TFCI values used for E-AGCH measurement in manual mode.

param etfci Range: 0 to 127

set_mode(*mode: RsCmwWcdmaSig.enums.AutoManualMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EAGCh:ETFCi:MODE
driver.configure.eagch.etfci.set_mode(mode = enums.AutoManualMode.AUTO)
```

Specifies the mode of expected E-TFCI selection.

param mode AUTO | MANual Automatic according to AG pattern or manual

7.1.20 Ehich

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:EHICh:TOUT
CONFigure:WCDMa:SIGNaling<Instance>:EHICh:REPetition
CONFigure:WCDMa:SIGNaling<Instance>:EHICh:MFRames
CONFigure:WCDMa:SIGNaling<Instance>:EHICh:LIMit
```

class Ehich

Ehich commands group definition. 5 total commands, 1 Sub-groups, 4 group commands

get_limit() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:LIMit
value: float = driver.configure.ehich.get_limit()
```

Specifies limits for the results of the E-HICH measurement.

return false_ratio: Upper limit for E-HICH reception 'False Ratio' result Range: 0 % to 100 %, Unit: %

get_mframes() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:MFRames
value: int = driver.configure.ehich.get_mframes()
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) .

return meas_frames: Range: 100 to 1E+6

get_repetition() → RsCmwWcdmaSig.enums.Repeat

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:REPetition
value: enums.Repeat = driver.configure.ehich.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Ehich.mframes to define the number of subframes to be measured per single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:TOUT
value: float = driver.configure.ehich.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_limit(*false_ratio: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:LIMit
driver.configure.ehich.set_limit(false_ratio = 1.0)
```

Specifies limits for the results of the E-HICH measurement.

param false_ratio Upper limit for E-HICH reception 'False Ratio' result Range: 0 % to 100 %, Unit: %

set_mframes(*meas_frames: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:MFRames
driver.configure.ehich.set_mframes(meas_frames = 1)
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) .

param meas_frames Range: 100 to 1E+6

set_repetition(*repetition: RsCmwWcdmaSig.enums.Repeat*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:REPetition
driver.configure.ehich.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Ehich.mframes to define the number of subframes to be measured per single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:TOUT
driver.configure.ehich.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

Cloning the Group

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

Subgroups

7.1.20.1 Smode

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:EHICh:SMODe:AVERage
```

class Smode

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

get_average() → RsCmwWcdmaSig.enums.AveragingMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:SMODe:AVERage
value: enums.AveragingMode = driver.configure.ehich.smode.get_average()
```

Specifies calculation algorithm for average statistics. Only remote command is provided. No corresponding manual setting is existing in the GUI. The setting is especially useful if 'Repetition' = 'Continuous'.

return mode: WINDow | CONTinuous WINDow: average results calculated per statistic cycle ('Measure Frames') CONTinuous: average results calculated since the beginning of the measurement

set_average(*mode: RsCmwWcdmaSig.enums.AveragingMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:EHICh:SMODe:AVERage
driver.configure.ehich.smode.set_average(mode = enums.AveragingMode.CONTinuous)
```

Specifies calculation algorithm for average statistics. Only remote command is provided. No corresponding manual setting is existing in the GUI. The setting is especially useful if 'Repetition' = 'Continuous'.

param mode WINDow | CONTinuous WINDow: average results calculated per statistic cycle ('Measure Frames') CONTinuous: average results calculated since the beginning of the measurement

7.1.21 Ergch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:TOUT
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:REPetition
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:MFRames
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:LIMit
```

class Ergch

Ergch commands group definition. 9 total commands, 1 Sub-groups, 4 group commands

class LimitStruct

Structure for reading output parameters. Fields:

- Missed_Down_Ratio: float: Range: 0 % to 100 %
- Missed_Up_Ratio: float: Range: 0 % to 100 %
- Missed_Hold_Ratio: float: Range: 0 % to 100 %

get_limit() → LimitStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:LIMit
value: LimitStruct = driver.configure.ergch.get_limit()
```

Specifies the upper limit for the missed DOWN / UP / HOLD ratios.

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

get_mframes() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:MFRames
value: int = driver.configure.ergch.get_mframes()
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) . Ideally, one relative grant value is detected per TTI.

return meas_frames: Range: 1 to 1E+6

get_repetition() → RsCmwWcdmaSig.enums.Repeat

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:REPetition
value: enums.Repeat = driver.configure.ergch.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Ergch.mframes to define the number of subframes to be measured per single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:TOUT
value: float = driver.configure.ergch.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: No help available

set_limit(*value: RsCmwWcdmaSig.Implementations.Configure_.Ergch.Ergch.LimitStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:LIMit
driver.configure.ergch.set_limit(value = LimitStruct())
```

Specifies the upper limit for the missed DOWN / UP / HOLD ratios.

param value see the help for LimitStruct structure arguments.

set_mframes(*meas_frames: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:MFRames
driver.configure.ergch.set_mframes(meas_frames = 1)
```

Defines the number of subframes to be measured per measurement cycle (statistics cycle) . Ideally, one relative grant value is detected per TTI.

param meas_frames Range: 1 to 1E+6

set_repetition(*repetition: RsCmwWcdmaSig.enums.Repeat*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:REPetition
driver.configure.ergch.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwWcdmaSig.Configure.Ergch.mframes to define the number of subframes to be measured per single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:TOUT
driver.configure.ergch.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 No help available

Cloning the Group

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

Subgroups

7.1.21.1 Etfci

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:ETFCi:MODE
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:ETFCi:EXPected
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:ETFCi:INITial
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:ETFCi:MANual
CONFigure:WCDMa:SIGNaling<Instance>:ERGCh:ETFCi:AUTO
```

class Etfci

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

get_auto() → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:AUTO
value: List[int] = driver.configure.ergch.etfci.get_auto()
```

Queries the n calculated E-TFCI values according to the AG configuration. Number of values n is set by method RsCmwWcdmaSig.Configure.Ergch.Etfci.expected

return etfci: Range: 0 to 127

get_expected() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:EXPected
value: int = driver.configure.ergch.etfci.get_expected()
```

Specifies the number of valid E-TFCI values in the expected E-TFCI table, see also method RsCmwWcdmaSig.Configure.Ergch. Etfci.manual method RsCmwWcdmaSig.Configure.Ergch.Etfci.auto

return no_expected: Range: 3 to 11

get_initial() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:INITial
value: int = driver.configure.ergch.etfci.get_initial()
```

Position of the initial operating point in the expected E-TFCI table. If the operating point of the UE is shifted outside the E-TFCI range, the initial operating point is readjusted. See also: method RsCmwWcdmaSig.Configure.Ergch.Etfci.manual and method RsCmwWcdmaSig.Configure.Ergch.Etfci.auto

return index: Range: 2 to (No. of expected ETFCI) - 1

get_manual() → List[int]

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:MANual
value: List[int] = driver.configure.ergch.etfci.get_manual()
```

Specifies the n E-TFCI values set manually. Number of values n is set by method RsCmwWcdmaSig.Configure.Ergch.Etfci. expected

return etfci: Range: 0 to 127

get_mode() → RsCmwWcdmaSig.enums.AutoManualMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:MODE
value: enums.AutoManualMode = driver.configure.ergch.etfci.get_mode()
```

Specifies the mode of expected E-TFCI selection.

return mode: AUTO | MANual Automatic according to AG pattern or manual

set_expected(*no_expected: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:EXPected
driver.configure.ergch.etfci.set_expected(no_expected = 1)
```

Specifies the number of valid E-TFCI values in the expected E-TFCI table, see also method RsCmwWcdmaSig.Configure.Ergch. Etfci.manual method RsCmwWcdmaSig.Configure.Ergch.Etfci.auto

param no_expected Range: 3 to 11

set_initial(*index: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:INITial
driver.configure.ergch.etfci.set_initial(index = 1)
```

Position of the initial operating point in the expected E-TFCI table. If the operating point of the UE is shifted outside the E-TFCI range, the initial operating point is readjusted. See also: method RsCmwWcdmaSig.Configure.Ergch.Etfci.manual and method RsCmwWcdmaSig.Configure.Ergch.Etfci.auto

param index Range: 2 to (No. of expected ETFCI) - 1

set_manual(*etfci: List[int]*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:MANual
driver.configure.ergch.etfci.set_manual(etfci = [1, 2, 3])
```

Specifies the n E-TFCI values set manually. Number of values n is set by method RsCmwWcdmaSig.Configure.Ergch.Etfci. expected

param etfci Range: 0 to 127

set_mode(*mode: RsCmwWcdmaSig.enums.AutoManualMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:ERGCh:ETFCi:MODE
driver.configure.ergch.etfci.set_mode(mode = enums.AutoManualMode.AUTO)
```

Specifies the mode of expected E-TFCI selection.

param mode AUTO | MANual Automatic according to AG pattern or manual

7.1.22 Sms

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:SMS:KTLoop
```

class Sms

Sms commands group definition. 20 total commands, 2 Sub-groups, 1 group commands

get_kt_loop() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:KTLoop
value: bool = driver.configure.sms.get_kt_loop()
```

Specifies whether the test loop is kept closed for an established RMC connection with test loop, when an SMS message is sent to the UE.

return enable: OFF | ON

set_kt_loop(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:KTLoop
driver.configure.sms.set_kt_loop(enable = False)
```

Specifies whether the test loop is kept closed for an established RMC connection with test loop, when an SMS message is sent to the UE.

param enable OFF | ON

Cloning the Group

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

Subgroups

7.1.22.1 Outgoing

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:BINary
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:PIDentifier
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:DCODing
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:CGRoup
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:MCLass
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:OSADdress
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:OADDress
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:RMCDelay
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:LHANdling
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:MESHandling
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:INTernal
```

class Outgoing

Outgoing commands group definition. 17 total commands, 2 Sub-groups, 12 group commands

get_binary() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:BINary
value: float = driver.configure.sms.outgoing.get_binary()
```

Defines the SMS message encoded as 8-bit binary data.

return sms_binary: SMS message in hexadecimal format.

get_cgroup() → RsCmwWcdmaSig.enums.CodingGroup

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:CGRoup
value: enums.CodingGroup = driver.configure.sms.outgoing.get_cgroup()
```

Defines how to interpret SMS signaling information. Coding groups are defined in 3GPP TS 23.038 chapter 4.

return coding_group: GDCoding | DCMClass | REServed GDCoding: general data coding DCMClass: data coding / message class REServed: reserved coding groups

get_dcoding() → RsCmwWcdmaSig.enums.SmsDataCoding

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:DCODing
value: enums.SmsDataCoding = driver.configure.sms.outgoing.get_dcoding()
```

Defines the short message coding.

return data_coding: BIT7 | BIT8 | REServed BIT7: GSM 7-bit default alphabet BIT8: 8-bit data for SMS binary REServed: reserved character set

get_internal() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:INTernal
value: str = driver.configure.sms.outgoing.get_internal()
```

Defines the message text for SMS messages to be sent to the UE. It is encoded as 7-bit ASCII text.

return sms_internal: String with up to 800 characters

get_lhandling() → RsCmwWcdmaSig.enums.LongSmsHandling

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:LHANdling
value: enums.LongSmsHandling = driver.configure.sms.outgoing.get_lhandling()
```

Defines the handling of an SMS message exceeding 160 characters.

return lsms_handling: TRUNcate | MSMS TRUNcate: truncate MSMS: multiple SMS

get_mclass() → RsCmwWcdmaSig.enums.MessageClass

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:MCLass
value: enums.MessageClass = driver.configure.sms.outgoing.get_mclass()
```

Specifies default routing of SMS as defined in 3GPP TS 23.038. The UEs override the default routing by selecting their own routing.

return message_class: CL0 | CL1 | CL2 | CL3 | NONE CL0: class 0, SMS not to be stored automatically CL1: SMS to be stored in mobile equipment CL2: SMS to be stored in (U) SIM CL3: SMS to be stored in terminal equipment (see 3GPP TS 27.005) NONE: no message class (relevant only for general data coding)

get_mes_handling() → RsCmwWcdmaSig.enums.MessageHandling

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:MESHandling
value: enums.MessageHandling = driver.configure.sms.outgoing.get_mes_handling()
```

Specifies whether the outgoing message text is entered manually via method RsCmwWcdmaSig.Configure.Sms.Outgoing.internal or an existing SMS file is used. The SMS file is selected via method RsCmwWcdmaSig.Configure.Sms.Outgoing.File.value.

return message_handling: INTernal | FILE INTernal: content entered manually FILE: specified *.sms file is used

get_oaddress() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:OADDress
value: str = driver.configure.sms.outgoing.get_oaddress()
```

Specifies the phone number of the device which has sent SMS.

return orig_address: No help available

get_os_address() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:OSADdress
value: str = driver.configure.sms.outgoing.get_os_address()
```

Specifies the phone number of SMS center.

return orig_smsca_ddress: No help available

get_pidentifier() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:PIDentifier
value: float = driver.configure.sms.outgoing.get_pidentifier()
```

Specifies the SMS protocol identifier.

return idn: Range: 0 to 255

get_rmc_delay() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:RMCDelay
value: float or bool = driver.configure.sms.outgoing.get_rmc_delay()
```

Defines the time between sending of an SMS message and re-establishment of the RMC connection.

return delay: Range: 1 s to 5 s Additional parameters: OFF | ON (disables the delay | enables the delay using the previous/default value)

get_udheader() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
value: float or bool = driver.configure.sms.outgoing.get_udheader()
```

Configures the TP user data header.

return header: Up to 16 hexadecimal digits Range: #H0 to #HFFFFFFFFFFFFFFFF Additional parameters: OFF | ON (disables | enables sending the header)

set_binary(*sms_binary: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:BINary
driver.configure.sms.outgoing.set_binary(sms_binary = 1.0)
```

Defines the SMS message encoded as 8-bit binary data.

param sms_binary SMS message in hexadecimal format.

set_cgroup(*coding_group: RsCmwWcdmaSig.enums.CodingGroup*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:CGRoup
driver.configure.sms.outgoing.set_cgroup(coding_group = enums.CodingGroup.
↪DCMClass)
```

Defines how to interpret SMS signaling information. Coding groups are defined in 3GPP TS 23.038 chapter 4.

param coding_group GDCoding | DCMClass | REServed GDCoding: general data coding DCMClass: data coding / message class REServed: reserved coding groups

set_dcoding(*data_coding: RsCmwWcdmaSig.enums.SmsDataCoding*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:DCODing
driver.configure.sms.outgoing.set_dcoding(data_coding = enums.SmsDataCoding.
↪BIT7)
```

Defines the short message coding.

param data_coding BIT7 | BIT8 | REServed BIT7: GSM 7-bit default alphabet BIT8: 8-bit data for SMS binary REServed: reserved character set

set_internal(*sms_internal: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:INTernal
driver.configure.sms.outgoing.set_internal(sms_internal = '1')
```

Defines the message text for SMS messages to be sent to the UE. It is encoded as 7-bit ASCII text.

param sms_internal String with up to 800 characters

set_lhandling(*lsms_handling: RsCmwWcdmaSig.enums.LongSmsHandling*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:LHANdling
driver.configure.sms.outgoing.set_lhandling(lsms_handling = enums.
↪LongSmsHandling.MSMS)
```

Defines the handling of an SMS message exceeding 160 characters.

param lsms_handling TRUNcate | MSMS TRUNcate: truncate MSMS: multiple SMS

set_mclass(*message_class: RsCmwWcdmaSig.enums.MessageClass*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:MCLass
driver.configure.sms.outgoing.set_mclass(message_class = enums.MessageClass.CL0)
```

Specifies default routing of SMS as defined in 3GPP TS 23.038. The UEs override the default routing by selecting their own routing.

param message_class CL0 | CL1 | CL2 | CL3 | NONE CL0: class 0, SMS not to be stored automatically CL1: SMS to be stored in mobile equipment CL2: SMS to be stored in (U) SIM CL3: SMS to be stored in terminal equipment (see 3GPP TS 27.005) NONE: no message class (relevant only for general data coding)

set_mes_handling(*message_handling: RsCmwWcdmaSig.enums.MessageHandling*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:MESHandling
driver.configure.sms.outgoing.set_mes_handling(message_handling = enums.
↪MessageHandling.FILE)
```

Specifies whether the outgoing message text is entered manually via method RsCmwWcdmaSig.Configure.Sms.Outgoing.internal or an existing SMS file is used. The SMS file is selected via method RsCmwWcdmaSig.Configure.Sms.Outgoing.File.value.

param message_handling INTernal | FILE INTernal: content entered manually FILE: specified *.sms file is used

set_oaddress(*orig_address: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:OADDress
driver.configure.sms.outgoing.set_oaddress(orig_address = '1')
```

Specifies the phone number of the device which has sent SMS.

param orig_address No help available

set_os_address(*orig_smsca_ddress: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:OSADdress
driver.configure.sms.outgoing.set_os_address(orig_smsca_ddress = '1')
```

Specifies the phone number of SMS center.

param orig_smsca_ddress No help available

set_pidentifier(*idn: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:PIDentifier
driver.configure.sms.outgoing.set_pidentifier(idn = 1.0)
```

Specifies the SMS protocol identifier.

param idn Range: 0 to 255

set_rmc_delay(*delay: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:RMCDelay
driver.configure.sms.outgoing.set_rmc_delay(delay = 1.0)
```

Defines the time between sending of an SMS message and re-establishment of the RMC connection.

param delay Range: 1 s to 5 s Additional parameters: OFF | ON (disables the delay | enables the delay using the previous/default value)

set_udheader(*header: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
driver.configure.sms.outgoing.set_udheader(header = 1.0)
```

Configures the TP user data header.

param header Up to 16 hexadecimal digits Range: #H0 to #HFFFFFFFFFFFFFFFF
Additional parameters: OFF | ON (disables | enables sending the header)

Cloning the Group

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

Subgroups

7.1.22.1.1 SctStamp

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSOurce
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DATE
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME
```

class SctStamp

SctStamp commands group definition. 3 total commands, 0 Sub-groups, 3 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_date() → DateStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:SCTStamp:DATE
value: DateStruct = driver.configure.sms.outgoing.sctStamp.get_date()
```

Specifies the service center time stamp date for the time source DATE (see method RsCmwWcdmaSig.Configure.Sms.Outgoing. SctStamp.tsource) .

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

get_time() → TimeStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:SCTStamp:TIME
value: TimeStruct = driver.configure.sms.outgoing.sctStamp.get_time()
```

Specifies the service center time stamp time for the time source DATE (see method RsCmwWcdmaSig.Configure.Sms.Outgoing. SctStamp.tsource) .

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

`get_tsource()` → RsCmwWcdmaSig.enums.SourceTime

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:SCTStamp:TSOurce
value: enums.SourceTime = driver.configure.sms.outgoing.sctStamp.get_tsource()
```

Selects the date and time source for service center time stamp. INTRO_CMD_HELP: The time source 'DATE' is configured via the following commands:

- method RsCmwWcdmaSig.Configure.Sms.Outgoing.SctStamp.date
- method RsCmwWcdmaSig.Configure.Sms.Outgoing.SctStamp.time

return source_time: CMWTime | DATE CMWTime: Windows date and time DATE: Date and time specified via remote commands

`set_date`(*value: RsCmwWcdmaSig.Implementations.Configure_.Sms_.Outgoing_.SctStamp.SctStamp.DateStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:SCTStamp:DATE
driver.configure.sms.outgoing.sctStamp.set_date(value = DateStruct())
```

Specifies the service center time stamp date for the time source DATE (see method RsCmwWcdmaSig.Configure.Sms.Outgoing. SctStamp.tsource) .

param value see the help for DateStruct structure arguments.

`set_time`(*value: RsCmwWcdmaSig.Implementations.Configure_.Sms_.Outgoing_.SctStamp.SctStamp.TimeStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:SCTStamp:TIME
driver.configure.sms.outgoing.sctStamp.set_time(value = TimeStruct())
```

Specifies the service center time stamp time for the time source DATE (see method RsCmwWcdmaSig.Configure.Sms.Outgoing. SctStamp.tsource) .

param value see the help for TimeStruct structure arguments.

`set_tsource`(*source_time: RsCmwWcdmaSig.enums.SourceTime*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:SCTStamp:TSOurce
driver.configure.sms.outgoing.sctStamp.set_tsource(source_time = enums.
↪SourceTime.CMWTime)
```

Selects the date and time source for service center time stamp. INTRO_CMD_HELP: The time source 'DATE' is configured via the following commands:

- method RsCmwWcdmaSig.Configure.Sms.Outgoing.SctStamp.date

- method RsCmwWcdmaSig.Configure.Sms.Outgoing.SctStamp.time

  param source_time CMWTime | DATE CMWTime: Windows date and time DATE: Date and time specified via remote commands

7.1.22.1.2 File

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:FILE:INFO
CONFigure:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:FILE
```

class File

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

class InfoStruct

Structure for reading output parameters. Fields:

- Reserved: str: For future use
- Message_Encoding: str: Encoding of the ANSI message (ASCII, binary, Unicode)
- Message_Text: str: Message text
- Message_Length: int: The number of characters in the message Range: 0 to 10E+3

get_info() → InfoStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:FILE:INFO
value: InfoStruct = driver.configure.sms.outgoing.file.get_info()
```

Display information of the outgoing message file referenced by method RsCmwWcdmaSig.Configure.Sms.Outgoing.File.value.

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

get_value() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:FILE
value: str = driver.configure.sms.outgoing.file.get_value()
```

Selects an outgoing message file. To view details of the message use method RsCmwWcdmaSig.Configure.Sms.Outgoing.File. info. The message files are stored in the directory D:/Rohde-Schwarz/CMW/Data/SMS/WCDMA/....

return sms_file: Outgoing SMS file

set_value(*sms_file: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:OUTGoing:FILE
driver.configure.sms.outgoing.file.set_value(sms_file = '1')
```

Selects an outgoing message file. To view details of the message use method RsCmwWcdmaSig.Configure.Sms.Outgoing.File. info. The message files are stored in the directory D:/Rohde-Schwarz/CMW/Data/SMS/WCDMA/....

param sms_file Outgoing SMS file

7.1.22.2 Incoming

class Incoming
Incoming 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.sms.incoming.clone()
```

Subgroups

7.1.22.2.1 File

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:SMS:INComing:FILE:INFO
CONFigure:WCDMa:SIGNaling<Instance>:SMS:INComing:FILE
```

class File
File commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class InfoStruct
Structure for reading output parameters. Fields:

- Timestamp: str: Time stamp of sending
- Reserved: str: For future use
- Message_Encoding: str: Encoding of the ANSI message (ASCII, binary, Unicode)
- Message_Text: str: Message text
- Message_Length: int: The number of characters in the message Range: 0 to 10E+3
- Message_Segments: int: The segment number Range: 0 to 1000
- Used_Send_Method: enums.UsedSendMethod: WDEFault The send method used by the UE

get_info() → InfoStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:INComing:FILE:INFO
value: InfoStruct = driver.configure.sms.incoming.file.get_info()
```

Display information on the received message file referenced by method RsCmwWcdmaSig.Configure.Sms.Incoming.File.value.

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

get_value() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:INComing:FILE
value: str = driver.configure.sms.incoming.file.get_value()
```

Selects a received message file. The message files are stored in the directory D:/Rohde-Schwarz/CMW/Data/SMS/WCDMA/Received/.

return sms_file: String parameter to specify the received message file.

set_value(*sms_file: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:SMS:INComing:FILE
driver.configure.sms.incoming.file.set_value(sms_file = '1')
```

Selects a received message file. The message files are stored in the directory D:/Rohde-Schwarz/CMW/Data/SMS/WCDMA/Received/.

param sms_file String parameter to specify the received message file.

7.1.23 Cbs

class Cbs

Cbs commands group definition. 23 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.23.1 Ctch

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CBS:CTCH:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CBS:CTCH:PERiod
CONFigure:WCDMa:SIGNaling<Instance>:CBS:CTCH:FOFFset
CONFigure:WCDMa:SIGNaling<Instance>:CBS:CTCH:FMPLength
```

class Ctch

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

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:ENABle
value: bool = driver.configure.cbs.ctch.get_enable()
```

Enables CBS generally.

return enable: OFF | ON

get_fmp_length() → RsCmwWcdmaSig.enums.TtiExtended

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:FMPLength
value: enums.TtiExtended = driver.configure.cbs.ctch.get_fmp_length()
```

No command help available

return tti: No help available

get_foffset() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:FOFFset
value: int = driver.configure.cbs.ctch.get_foffset()
```

Offset (K) used for CTCH allocation within CTCH allocation period N, see method RsCmwWcdmaSig.Configure.Cbs.Ctch.period.

return frame_offset: The S-CCPCH TTI number, with the first CTCH allocated for cell broadcast. Range: 0 to N-1, Unit: frames

get_period() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:PERiod
value: int = driver.configure.cbs.ctch.get_period()
```

Specifies the periodicity of CTCH allocation within S-CCPCH.

return period: Duration of period (N) Range: 1 to 256, Unit: frames

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:ENABle
driver.configure.cbs.ctch.set_enable(enable = False)
```

Enables CBS generally.

param enable OFF | ON

set_fmp_length(*tti: RsCmwWcdmaSig.enums.TtiExtended*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:FMPLength
driver.configure.cbs.ctch.set_fmp_length(tti = enums.TtiExtended.M10)
```

No command help available

param tti No help available

set_foffset(*frame_offset: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:FOFFset
driver.configure.cbs.ctch.set_foffset(frame_offset = 1)
```

Offset (K) used for CTCH allocation within CTCH allocation period N, see method RsCmwWcdmaSig.Configure.Cbs.Ctch.period.

param frame_offset The S-CCPCH TTI number, with the first CTCH allocated for cell broadcast. Range: 0 to N-1, Unit: frames

set_period(*period: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:CTCH:PERiod
driver.configure.cbs.ctch.set_period(period = 1)
```

Specifies the periodicity of CTCH allocation within S-CCPCH.

param period Duration of period (N) Range: 1 to 256, Unit: frames

7.1.23.2 Drx

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CBS:DRX:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CBS:DRX:PERiod
CONFigure:WCDMa:SIGNaling<Instance>:CBS:DRX:LENGth
CONFigure:WCDMa:SIGNaling<Instance>:CBS:DRX:OFFSet
CONFigure:WCDMa:SIGNaling<Instance>:CBS:DRX:FEMPty
```

class Drx

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

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:ENABle
value: bool = driver.configure.cbs.drx.get_enable()
```

Enables DRX for CBS.

return enable: OFF | ON

get_fempty() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:FEMPty
value: bool = driver.configure.cbs.drx.get_fempty()
```

Specifies the handling of unused CTCH TTIs allocated for CBS.

return enable: OFF | ON OFF: no action for unused CTCH ON: fill unused CTCH with scheduling message

get_length() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:LENGth
value: int = driver.configure.cbs.drx.get_length()
```

Specifies the length of DRX (L) that the UE can use for the processing of particular CB message. P denotes the period of scheduling message, see method RsCmwWcdmaSig.Configure.Cbs.Drx.period. Define value matching with the position of the specific CB message within the CBS scheduling period.

return length_of_period: Range: 1 TTI to P-1 TTIs, Unit: TTI

get_offset() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:OFFSet
value: int = driver.configure.cbs.drx.get_offset()
```

Offset (O) within period of scheduling message (P) . This offset is used for the transmission of a scheduling message. See also: method RsCmwWcdmaSig.Configure.Cbs.Drx.period.

return offset: Range: 1 TTI to P-1 TTIs, Unit: TTI

get_period() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:PERiod
value: int or bool = driver.configure.cbs.drx.get_period()
```

Specifies the periodicity of DRX the UE can use for the processing of the CB message.

return period: Duration of period (P) Range: 1 to 256, Unit: TTIs Additional OFF | ON disables | enables the DRX period

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:ENABle
driver.configure.cbs.drx.set_enable(enable = False)
```

Enables DRX for CBS.

param enable OFF | ON

set_fempty(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:FEMPty
driver.configure.cbs.drx.set_fempty(enable = False)
```

Specifies the handling of unused CTCH TTIs allocated for CBS.

param enable OFF | ON OFF: no action for unused CTCH ON: fill unused CTCH with scheduling message

set_length(*length_of_period: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:LENGth
driver.configure.cbs.drx.set_length(length_of_period = 1)
```

Specifies the length of DRX (L) that the UE can use for the processing of particular CB message. P denotes the period of scheduling message, see method RsCmwWcdmaSig.Configure.Cbs.Drx.period. Define value matching with the position of the specific CB message within the CBS scheduling period.

param length_of_period Range: 1 TTI to P-1 TTIs, Unit: TTI

set_offset(*offset: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:OFFSet
driver.configure.cbs.drx.set_offset(offset = 1)
```

Offset (O) within period of scheduling message (P) . This offset is used for the transmission of a scheduling message. See also: method RsCmwWcdmaSig.Configure.Cbs.Drx.period.

param offset Range: 1 TTI to P-1 TTIs, Unit: TTI

set_period(*period: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:DRX:PERiod
driver.configure.cbs.drx.set_period(period = 1)
```

Specifies the periodicity of DRX the UE can use for the processing of the CB message.

param period Duration of period (P) Range: 1 to 256, Unit: TTIs Additional OFF | ON disables | enables the DRX period

7.1.23.3 Message

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:ID
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:IDTYpe
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:SERial
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:CGRoup
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:CATegory
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:SOURce
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:DATA
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:PERiod
```

class Message

Message commands group definition. 14 total commands, 3 Sub-groups, 9 group commands

class SerialStruct

Structure for reading output parameters. Fields:

- Geo_Scope: enums.GeoScope: CIMMediate | PLMN | SERVice | CNORmal The geographical area over which the message code is unique. CIMMediate: cell-wide, immediate display PLMN: PLMN wide SERVice: service area wide CNORmal: cell-wide, normal display
- Message_Code: int: CB message identification Range: 0 to 1023
- Auto_Incr: bool: OFF | ON OFF: no increase of UpdateNumber upon a CB message change ON: increase UpdateNumber automatically upon a CB message change
- Update_Number: int: Indication of a content change of the same CB message Range: 0 to 15

get_category() → RsCmwWcdmaSig.enums.Priority

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:CATegory
value: enums.Priority = driver.configure.cbs.message.get_category()
```

Indicates the privilege category of a CB message.

return category: BACKground | NORMal | HIGH BACKground: to be broadcast, when no CB messages of category high priority or normal are broadcast NORMal: to be

broadcast according to the associated repetition period HIGH: to be broadcast at the earliest opportunity

get_cgroup() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:CGRoup
value: int = driver.configure.cbs.message.get_cgroup()
```

Queries the coding group to be indicated to the CB message recipient. The coding group is defined in 3GPP TS 23. 038, section 5 as bits 4 to 7 of CBS data coding scheme.

return coding_group: 0: used for internal messages ('Data Source' = 'Use Internal') 1: used for CBS files (only language = 1: UCS2 is supported) Range: 0 to 1

get_data() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:DATA
value: str = driver.configure.cbs.message.get_data()
```

Defines the CB message text.

return data: Up to 1395 characters

get_enable() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ENABle
value: bool = driver.configure.cbs.message.get_enable()
```

Enables the particular CB message.

return enable: OFF | ON

get_id() → int

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ID
value: int = driver.configure.cbs.message.get_id()
```

Identifies source/type of a CB message. Edit this parameter for user-defined settings. Hexadecimal values are displayed for information.

return idn: Range: 0 to 65.535E+3

get_id_type() → RsCmwWcdmaSig.enums.CbsMessageSeverity

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:IDTYpe
value: enums.CbsMessageSeverity = driver.configure.cbs.message.get_id_type()
```

Specifies the severity of the CBS message ID as either decimal or hexadecimal number.

return type_py: UDEFined | APResidentia | AEXTreme | ASEVere | AAMBer | EARThquake | TSUNami | ETWarning | ETWTest UDEFined: user defined APResidentia: presidential level alerts (IDs 4370 and 4383) AEXTreme: extreme alerts (IDs 4371 to 4372 and 4384 to 4385) ASEVere: severe alerts (IDs 4373 to 4378 and 4386 to 4391) AAMBer: amber alerts (IDs 4379 and 4392) EARThquake: earthquake warning (ID

4352) TSUNami: tsunami warning (ID 4353) ETWarning: earthquake and tsunami warning (ID 4354) ETWTest: ETWS test message (ID 4355)

get_period() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:PERiod
value: float = driver.configure.cbs.message.get_period()
```

Repetition period to broadcast the CB message again.

return interval: Range: 1 s to 4096 s

get_serial() → SerialStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:SERial
value: SerialStruct = driver.configure.cbs.message.get_serial()
```

Specifies the unique CB message identification.

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

get_source() → RsCmwWcdmaSig.enums.MessageHandling

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:SOURce
value: enums.MessageHandling = driver.configure.cbs.message.get_source()
```

Specifies whether the CB message text is entered manually via method RsCmwWcdmaSig.Configure.Cbs.Message. data or an existing CBS file is used. The CBS file is selected via method RsCmwWcdmaSig.Configure.Cbs.Message.File.value.

return message_handling: INTernal | FILE INTernal: content entered manually FILE: specified *.cbs file is used

set_category(*category: RsCmwWcdmaSig.enums.Priority*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:CATegory
driver.configure.cbs.message.set_category(category = enums.Priority.BACKground)
```

Indicates the privilege category of a CB message.

param category BACKground | NORMal | HIGH BACKground: to be broadcast, when no CB messages of category high priority or normal are broadcast NORMal: to be broadcast according to the associated repetition period HIGH: to be broadcast at the earliest opportunity

set_data(*data: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:DATA
driver.configure.cbs.message.set_data(data = '1')
```

Defines the CB message text.

param data Up to 1395 characters

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ENABle
driver.configure.cbs.message.set_enable(enable = False)
```

Enables the particular CB message.

param enable OFF | ON

set_id(*idn: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ID
driver.configure.cbs.message.set_id(idn = 1)
```

Identifies source/type of a CB message. Edit this parameter for user-defined settings. Hexadecimal values are displayed for information.

param idn Range: 0 to 65.535E+3

set_id_type(*type_py: RsCmwWcdmaSig.enums.CbsMessageSeverity*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:IDTYpe
driver.configure.cbs.message.set_id_type(type_py = enums.CbsMessageSeverity.
↪AAMBer)
```

Specifies the severity of the CBS message ID as either decimal or hexadecimal number.

param type_py UDEFined | APResidentia | AEXTreme | ASEVere | AAMBer | EARThquake | TSUNami | ETWarning | ETWTest UDEFined: user defined APResidentia: presidential level alerts (IDs 4370 and 4383) AEXTreme: extreme alerts (IDs 4371 to 4372 and 4384 to 4385) ASEVere: severe alerts (IDs 4373 to 4378 and 4386 to 4391) AAMBer: amber alerts (IDs 4379 and 4392) EARThquake: earthquake warning (ID 4352) TSUNami: tsunami warning (ID 4353) ETWarning: earthquake and tsunami warning (ID 4354) ETWTest: ETWS test message (ID 4355)

set_period(*interval: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:PERiod
driver.configure.cbs.message.set_period(interval = 1.0)
```

Repetition period to broadcast the CB message again.

param interval Range: 1 s to 4096 s

set_serial(*value: RsCmwWcdmaSig.Implementations.Configure_.Cbs_.Message.Message.SerialStruct*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:SERial
driver.configure.cbs.message.set_serial(value = SerialStruct())
```

Specifies the unique CB message identification.

param value see the help for SerialStruct structure arguments.

set_source(*message_handling: RsCmwWcdmaSig.enums.MessageHandling*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:SOURce
driver.configure.cbs.message.set_source(message_handling = enums.
↪MessageHandling.FILE)
```

Specifies whether the CB message text is entered manually via method RsCmwWcdmaSig.Configure.Cbs.Message. data or an existing CBS file is used. The CBS file is selected via method RsCmwWcdmaSig.Configure.Cbs.Message.File.value.

param message_handling INTernal | FILE INTernal: content entered manually FILE: specified *.cbs file is used

Cloning the Group

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

Subgroups

7.1.23.3.1 Language

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:LANGuage
```

class Language

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

class GetStruct

Response structure. Fields:

- Language: int: Bits 0 to 3 of CBS data coding scheme Range: 0 to 15
- Lng_Indication: str: Language indication

get() → GetStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:LANGuage
value: GetStruct = driver.configure.cbs.message.language.get()
```

Specifies the language of CB message as defined in 3GPP TS 23.038.

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

set(*language: int*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:LANGuage
driver.configure.cbs.message.language.set(language = 1)
```

Specifies the language of CB message as defined in 3GPP TS 23.038.

param language Bits 0 to 3 of CBS data coding scheme Range: 0 to 15

7.1.23.3.2 File

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:FILE:INFO
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:FILE
```

class File

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

class InfoStruct

Structure for reading output parameters. Fields:

- Message_Encoding: str: Encoding of the CB message (UTF16)
- Message_Text: str: Message text
- Message_Length: int: The number of characters in the message Range: 0 to 600

get_info() → InfoStruct

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:FILE:INFO
value: InfoStruct = driver.configure.cbs.message.file.get_info()
```

Display information of the outgoing CB message file referenced by method RsCmwWcdmaSig.Configure.Cbs.Message.File.value.

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

get_value() → str

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:FILE
value: str = driver.configure.cbs.message.file.get_value()
```

Selects a CB message file. To view details of the message use method RsCmwWcdmaSig.Configure.Cbs.Message.File.info. The message files are stored in the directory D:/Rohde-Schwarz/CMW/Data/cbs/WCDMA/.

return file: File to be used for CB message

set_value(*file: str*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:FILE
driver.configure.cbs.message.file.set_value(file = '1')
```

Selects a CB message file. To view details of the message use method RsCmwWcdmaSig.Configure.Cbs.Message.File.info. The message files are stored in the directory D:/Rohde-Schwarz/CMW/Data/cbs/WCDMA/.

param file File to be used for CB message

7.1.23.3.3 Etws

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:ETWS:ALERt
CONFigure:WCDMa:SIGNaling<Instance>:CBS:MESSage:ETWS:POPup
```

class Etws

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

get_alert() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ETWS:ALERt
value: bool = driver.configure.cbs.message.etws.get_alert()
```

Deactivates / activates earthquake and tsunami warning system alerting.

return enable: OFF | ON

get_popup() → bool

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ETWS:POPup
value: bool = driver.configure.cbs.message.etws.get_popup()
```

Deactivates / activates earthquake and tsunami warning popup on display.

return enable: OFF | ON

set_alert(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ETWS:ALERt
driver.configure.cbs.message.etws.set_alert(enable = False)
```

Deactivates / activates earthquake and tsunami warning system alerting.

param enable OFF | ON

set_popup(*enable: bool*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:CBS:MESSage:ETWS:POPup
driver.configure.cbs.message.etws.set_popup(enable = False)
```

Deactivates / activates earthquake and tsunami warning popup on display.

param enable OFF | ON

7.1.24 Fading

class Fading
Fading commands group definition. 15 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.24.1 Carrier

class Carrier
Carrier commands group definition. 15 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.carrier.clone()
```

Subgroups

7.1.24.1.1 Fsimulator

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:STANdard
```

class Fsimulator
Fsimulator commands group definition. 9 total commands, 4 Sub-groups, 2 group commands

get_enable() → bool

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

Enables/disables the fading simulator.

return enable: OFF | ON

Global Repeated Capabilities: repcap.Carrier

get_standard() → RsCmwWcdmaSig.enums.FadingStandard

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:STANdard
value: enums.FadingStandard = driver.configure.fading.carrier.fsimulator.get_
↪standard()
```

Selects one of the propagation conditions defined in the annex B.2 of 3GPP TS 25.101.

return standard: C1 | C2 | C3 | C4 | C5 | C6 | C8 | PA3 | PB3 | VA3 | VA30 | VA12 | MPRopagation | BDEath | HST C1 to C6: case 1 to case 6 (multipath fading profile) C8: case 8 (for CQI test in multipath fading and HS-SCCH-less demodulation of HS-DSCH) PA3 | PB3: ITU PA3 / PB3 (multipath fading profile) VA3 | VA30 | VA12: ITU VA3 / VA30 / VA120 (multipath fading profile) MPRopagation: moving propagation BDEath: birth-death propagation HST: high-speed train

Global Repeated Capabilities: repcap.Carrier

set_enable(*enable: bool*) → None

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

Enables/disables the fading simulator.

param enable OFF | ON

Global Repeated Capabilities: repcap.Carrier

set_standard(*standard: RsCmwWcdmaSig.enums.FadingStandard*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:STANdard
driver.configure.fading.carrier.fsimulator.set_standard(standard = enums.
↪FadingStandard.B261)
```

Selects one of the propagation conditions defined in the annex B.2 of 3GPP TS 25.101.

param standard C1 | C2 | C3 | C4 | C5 | C6 | C8 | PA3 | PB3 | VA3 | VA30 | VA12 | MPRopagation | BDEath | HST C1 to C6: case 1 to case 6 (multipath fading profile) C8: case 8 (for CQI test in multipath fading and HS-SCCH-less demodulation of HS-DSCH) PA3 | PB3: ITU PA3 / PB3 (multipath fading profile) VA3 | VA30 | VA12: ITU VA3 / VA30 / VA120 (multipath fading profile) MPRopagation: moving propagation BDEath: birth-death propagation HST: high-speed train

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.24.1.1.1 Globale

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:GLOBal:SEED
```

class Globale

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

get_seed() → int

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

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

return seed: Range: 0 to 9

Global Repeated Capabilities: repcap.Carrier

set_seed(*seed: int*) → None

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

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

param seed Range: 0 to 9

Global Repeated Capabilities: repcap.Carrier

7.1.24.1.1.2 Restart

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:RESTart:MODE
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:RESTart
```

class Restart

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

get_mode() → RsCmwWcdmaSig.enums.AutoManualMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:RESTart:MODE
value: enums.AutoManualMode = driver.configure.fading.carrier.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 RsCmwWcdmaSig.Configure.Fading.Carrier.Fsimulator.Restart.set)

Global Repeated Capabilities: repcap.Carrier

set() → None

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

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

Global Repeated Capabilities: repcap.Carrier

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

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:RESTart:MODE
driver.configure.fading.carrier.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 RsCmwWcdmaSig.Configure.Fading.Carrier.Fsimulator.Restart.set)

Global Repeated Capabilities: repcap.Carrier

set_with_opc() → None

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

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

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

Global Repeated Capabilities: repcap.Carrier

7.1.24.1.1.3 Iloss

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:ILOSs:MODE
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:ILOSs:LOSS
```

class Iloss

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

get_loss() → float

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

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

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

Global Repeated Capabilities: repcap.Carrier

get_mode() → RsCmwWcdmaSig.enums.InsertLossMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:ILOSs:MODE
value: enums.InsertLossMode = driver.configure.fading.carrier.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

Global Repeated Capabilities: repcap.Carrier

set_loss(*insertion_loss: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:ILOSs:LOSS
driver.configure.fading.carrier.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 RsCmwWcdmaSig. Configure.Fading.Carrier.Fsimulator.Iloss.mode) .

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

Global Repeated Capabilities: repcap.Carrier

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

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:ILOSs:MODE
driver.configure.fading.carrier.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

Global Repeated Capabilities: repcap.Carrier

7.1.24.1.1.4 Dshift

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:DSHift:MODE
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:DSHift
```

class Dshift

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

get_mode() → RsCmwWcdmaSig.enums.InsertLossMode

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:DSHift:MODE
value: enums.InsertLossMode = driver.configure.fading.carrier.fsimulator.dshift.
↪get_mode()
```

Sets the Doppler shift mode.

return mode: NORMal | USER NORMal: the maximum Doppler frequency is determined by the fading profile USER: the maximum Doppler frequency can be adjusted manually

Global Repeated Capabilities: repcap.Carrier

get_value() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:DSHift
value: float = driver.configure.fading.carrier.fsimulator.dshift.get_value()
```

Displays the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwWcdmaSig.Configure.Fading.Carrier.Fsimulator.Dshift.mode) .

return frequency: Range: 1 Hz to 2000 Hz, Unit: Hz

Global Repeated Capabilities: repcap.Carrier

set_mode(*mode: RsCmwWcdmaSig.enums.InsertLossMode*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:DSHift:MODE
driver.configure.fading.carrier.fsimulator.dshift.set_mode(mode = enums.
↪InsertLossMode.NORMal)
```

Sets the Doppler shift mode.

param mode NORMal | USER NORMal: the maximum Doppler frequency is determined by the fading profile USER: the maximum Doppler frequency can be adjusted manually

Global Repeated Capabilities: repcap.Carrier

set_value(*frequency: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:DSHift
driver.configure.fading.carrier.fsimulator.dshift.set_value(frequency = 1.0)
```

Displays the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwWcdmaSig.Configure.Fading.Carrier.Fsimulator.Dshift.mode) .

param frequency Range: 1 Hz to 2000 Hz, Unit: Hz

Global Repeated Capabilities: repcap.Carrier

7.1.24.1.2 Awgn

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:AWGN:NOISe
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:AWGN:ENABle
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:AWGN:SNRatio
```

class Awgn

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

get_enable() → bool

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

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

return enable: OFF | ON

Global Repeated Capabilities: repcap.Carrier

get_noise() → float

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>:AWGN:NOISe
value: float = driver.configure.fading.carrier.awgn.get_noise()
```

Sets the total AWGN level within the channel bandwidth, applicable to AWGN inserted via the internal fading module. For multi-carrier scenarios, the same settings are applied to all carriers. Thus it is sufficient to configure one carrier.

return noise: Range: depends on connector, external attenuation, base level and insertion loss , Unit: dBm

Global Repeated Capabilities: repcap.Carrier

get_sn_ratio() → float

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

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

return ratio: Range: -50 dB to 40 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

set_enable(*enable: bool*) → None

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

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

param enable OFF | ON

Global Repeated Capabilities: repcap.Carrier

set_noise(*noise: float*) → None

```
# SCPI: CONFigure:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>:AWGN:NOISe
driver.configure.fading.carrier.awgn.set_noise(noise = 1.0)
```

Sets the total AWGN level within the channel bandwidth, applicable to AWGN inserted via the internal fading module. For multi-carrier scenarios, the same settings are applied to all carriers. Thus it is sufficient to configure one carrier.

param noise Range: depends on connector, external attenuation, base level and insertion loss , Unit: dBm

Global Repeated Capabilities: repcap.Carrier

7.1.24.1.3 Power

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:POWer:SUM
```

class Power

Power commands group definition. 3 total commands, 1 Sub-groups, 1 group commands

get_sum() → float

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

Queries the calculated total power (signal + noise) on the downlink carrier.

return power: Unit: dBm

Global Repeated Capabilities: repcap.Carrier

Cloning the Group

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

Subgroups

7.1.24.1.3.1 Noise

SCPI Commands

```
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:POWer:NOISe:TOTal
CONFigure:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:POWer:NOISe
```

class Noise

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

get_total() → float

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

Queries the total noise power.

return noise_power: Unit: dBm

Global Repeated Capabilities: repcap.Carrier

get_value() → float

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

Queries the calculated noise power on the downlink carrier.

return noise_power: Unit: dBm

Global Repeated Capabilities: repcap.Carrier

7.2 Prepare

class Prepare

Prepare commands group definition. 9 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.1 Handover

SCPI Commands

```
PREPare:WCDMa:SIGNaling<Instance>:HANDover:DESTination
PREPare:WCDMa:SIGNaling<Instance>:HANDover:MMODe
```

class Handover

Handover commands group definition. 9 total commands, 2 Sub-groups, 2 group commands

get_destination() → str

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:DESTination
value: str = driver.prepare.handover.get_destination()
```

Selects the handover destination. A complete list of all supported values can be displayed using method RsCmwWcdmaSig. Prepare.Handover.Catalog.destination.

return destination: Destination as string

get_mmode() → RsCmwWcdmaSig.enums.MobilityMode

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:MMODe
value: enums.MobilityMode = driver.prepare.handover.get_mmode()
```

Selects the mechanism to be used for mobility management.

return mobility_mode: HANDover | REDirection | CCORder Handover, redirection, or cell change order

set_destination(*destination: str*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:DESTination
driver.prepare.handover.set_destination(destination = '1')
```

Selects the handover destination. A complete list of all supported values can be displayed using method RsCmwWcdmaSig. Prepare.Handover.Catalog.destination.

param destination Destination as string

set_mmode(*mobility_mode: RsCmwWcdmaSig.enums.MobilityMode*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:MMODe
driver.prepare.handover.set_mmode(mobility_mode = enums.MobilityMode.CCORder)
```

Selects the mechanism to be used for mobility management.

param mobility_mode HANDover | REDirection | CCORder Handover, redirection, or cell change order

Cloning the Group

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

Subgroups

7.2.1.1 Catalog

SCPI Commands

```
PREPare:WCDMa:SIGNaling<Instance>:HANDover:CATalog:DESTination
```

class Catalog

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

get_destination() → List[str]

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:CATalog:DESTination
value: List[str] = driver.prepare.handover.catalog.get_destination()
```

Lists all handover destinations that can be selected using method RsCmwWcdmaSig.Prepare.Handover.destination.

return destination_list: Comma-separated list of all supported destinations. Each destination is represented as a string.

7.2.1.2 External

SCPI Commands

```
PREPare:WCDMa:SIGNaling<Instance>:HANDover:EXTernal:DESTination
PREPare:WCDMa:SIGNaling<Instance>:HANDover:EXTernal:LTE
PREPare:WCDMa:SIGNaling<Instance>:HANDover:EXTernal:GSM
PREPare:WCDMa:SIGNaling<Instance>:HANDover:EXTernal:CDMA
PREPare:WCDMa:SIGNaling<Instance>:HANDover:EXTernal:EVDO
PREPare:WCDMa:SIGNaling<Instance>:HANDover:EXTernal:WCDMa
```

class External

External commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

class CdmaStruct
Structure for reading output parameters. Fields:

- Band_Class: enums.BandClass: USC | KCEL | NAPC | TACS | JTAC | KPCS | N45T | IM2K | NA7C | B18M | NA8S | PA4M | PA8M | IEXT | USPC | AWS | U25B | U25F | NA9C | PS7C | LO7C 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'
- Dl_Channel: int: Channel number Range: 0 to 2108, depending on band class, see table below

class EvdoStruct
Structure for reading output parameters. Fields:

- Band_Class: enums.BandClass: USC | KCEL | NAPC | TACS | JTAC | KPCS | N45T | IM2K | NA7C | B18M | NA8S | PA4M | PA8M | IEXT | USPC | AWS | U25B | U25F | NA9C | PS7C | LO7C 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'
- Dl_Channel: int: Channel number Range: 0 to 2108, depending on band class, see table below

class GsmStruct
Structure for reading output parameters. Fields:

- Band: enums.GsmBand: G04 | G085 | G09 | G18 | G19 GSM 400, GSM 850, GSM 900, GSM 1800, GSM 1900
- Dl_Channel: int: Channel number used for the broadcast control channel (BCCH) Range: The allowed range depends on the operating band, see table below.

class LteStruct
Structure for reading output parameters. Fields:

- 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 | OB45 | OB46 | OB65 | OB66 | OB67 | OB252 | OB255 Operating band 1 to 46, 65 to 67, 252, 255
- Dl_Channel: int: Downlink channel number Range: The allowed range depends on the LTE band, see table below.

class WcdmaStruct
Structure for reading output parameters. Fields:

- Band: enums.OperationBand: OB1 | … | OB14 | OB19 | … | OB22 | OB25 | OB26 | OBS1 | … | OBS3 | OBL1 | UDEFined OB1, …, OB14: operating band I to XIV OB19, …, OB22: operating band XIX to XXII OB25, OB26: operating band XXV, XXVI OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L UDEFined: user defined

- Dl_Channel: int: For channel number ranges depending on operating bands see Table 'Operating bands for uplink signals'.

get_cdma() → CdmaStruct

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:CDMA
value: CdmaStruct = driver.prepare.handover.external.get_cdma()
```

Configure the destination parameters for handover to a CDMA2000 or 1xEV-DO destination at another instrument.

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

get_destination() → RsCmwWcdmaSig.enums.HoverExtDestination

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:DESTination
value: enums.HoverExtDestination = driver.prepare.handover.external.get_
↪destination()
```

Selects the target radio access technology for handover to another instrument.

return destination: WCDMa | GSM | LTE | EVDO | CDMA

get_evdo() → EvdoStruct

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:EVDO
value: EvdoStruct = driver.prepare.handover.external.get_evdo()
```

Configure the destination parameters for handover to a CDMA2000 or 1xEV-DO destination at another instrument.

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

get_gsm() → GsmStruct

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:GSM
value: GsmStruct = driver.prepare.handover.external.get_gsm()
```

Configures the destination parameters for handover to a GSM destination at another instrument.

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

get_lte() → LteStruct

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:LTE
value: LteStruct = driver.prepare.handover.external.get_lte()
```

No command help available

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

get_wcdma() → WcdmaStruct

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:WCDMa
value: WcdmaStruct = driver.prepare.handover.external.get_wcdma()
```

Configures the destination parameters for handover to a WCDMA destination at another instrument.

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

set_cdma(*value: RsCmwWcdmaSig.Implementations.Prepare_.Handover_.External.External.CdmaStruct*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:CDMA
driver.prepare.handover.external.set_cdma(value = CdmaStruct())
```

Configure the destination parameters for handover to a CDMA2000 or 1xEV-DO destination at another instrument.

param value see the help for CdmaStruct structure arguments.

set_destination(*destination: RsCmwWcdmaSig.enums.HoverExtDestination*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:DESTination
driver.prepare.handover.external.set_destination(destination = enums.
↪HoverExtDestination.CDMA)
```

Selects the target radio access technology for handover to another instrument.

param destination WCDMa | GSM | LTE | EVDO | CDMA

set_evdo(*value: RsCmwWcdmaSig.Implementations.Prepare_.Handover_.External.External.EvdoStruct*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:EVDO
driver.prepare.handover.external.set_evdo(value = EvdoStruct())
```

Configure the destination parameters for handover to a CDMA2000 or 1xEV-DO destination at another instrument.

param value see the help for EvdoStruct structure arguments.

set_gsm(*value: RsCmwWcdmaSig.Implementations.Prepare_.Handover_.External.External.GsmStruct*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:GSM
driver.prepare.handover.external.set_gsm(value = GsmStruct())
```

Configures the destination parameters for handover to a GSM destination at another instrument.

param value see the help for GsmStruct structure arguments.

set_lte(*value: RsCmwWcdmaSig.Implementations.Prepare_.Handover_.External.External.LteStruct*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:LTE
driver.prepare.handover.external.set_lte(value = LteStruct())
```

No command help available

param value see the help for LteStruct structure arguments.

set_wcdma(*value: RsCmwWcdmaSig.Implementations.Prepare_.Handover_.External.External.WcdmaStruct*) → None

```
# SCPI: PREPare:WCDMa:SIGNaling<instance>:HANDover:EXTernal:WCDMa
driver.prepare.handover.external.set_wcdma(value = WcdmaStruct())
```

Configures the destination parameters for handover to a WCDMA destination at another instrument.

param value see the help for WcdmaStruct structure arguments.

7.3 Sense

class Sense

Sense commands group definition. 64 total commands, 11 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.1 Elogging

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:ELOGging:LAST
SENSe:WCDMa:SIGNaling<Instance>:ELOGging:ALL
```

class Elogging

Elogging 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 CONTinue means the continuation of previous entry.
- Description: List[str]: Text string describing the event

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 CONTinue means the continuation of previous entry.

- Description: str: Text string describing the event

get_all() → AllStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:ELOGging:ALL
value: AllStruct = driver.sense.elogging.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:WCDMa:SIGNaling<instance>:ELOGging:LAST
value: LastStruct = driver.sense.elogging.get_last()
```

Queries the latest entry of the event log.

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

7.3.2 UeReport

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UEReport:CCELl
```

class UeReport

UeReport commands group definition. 5 total commands, 1 Sub-groups, 1 group commands

class CcellStruct

Structure for reading output parameters. Fields:

- Cpi_Ch_Rscp: List[float]: No parameter help available
- Cpi_Ch_Ec_No: List[float]: No parameter help available
- Tch_Bler: List[float]: No parameter help available
- Tx_Power: List[float]: No parameter help available
- Rx_Tx_Time_Diff: List[int]: No parameter help available
- Pathloss: float: No parameter help available

get_ccell() → CcellStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UEReport:CCELl
value: CcellStruct = driver.sense.ueReport.get_ccell()
```

Returns the UE measurement report contents for the current cell. See also 'UTRA FDD (Current Cell) '. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

Cloning the Group

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

Subgroups

7.3.2.1 Ncell

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UEReport:NCELl<DownCarrier>
```

class Ncell

Ncell commands group definition. 4 total commands, 3 Sub-groups, 1 group commands

class GetStruct

Response structure. Fields:

- Cpi_Ch_Rscp: List[float]: No parameter help available
- Cpi_Ch_Ec_No: List[float]: No parameter help available
- Rssi: List[float]: No parameter help available
- Sfn_Cfn_Time_Diff: List[int]: No parameter help available
- Pathloss: float: No parameter help available

get(*downCarrier*=<*DownCarrier.Dc1: 1*>) → GetStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UEReport:NCELl<nr>
value: GetStruct = driver.sense.ueReport.ncell.get(downCarrier = repcap.
↪DownCarrier.Dc1)
```

Returns the UE measurement report contents for additional carrier in multi-carrier operation. See also 'UTRA FDD (Carrier 2 / Carrier 3) '. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

param downCarrier optional repeated capability selector. Default value: Dc1

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.ueReport.ncell.clone()
```

Subgroups

7.3.2.1.1 Gsm

class Gsm
Gsm 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.ueReport.ncell.gsm.clone()
```

Subgroups

7.3.2.1.1.1 Cell

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UEReport:NCELl:GSM:CELL<Cell>
```

class Cell
Cell commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class GetStruct
Response structure. Fields:

- Rssi: List[float]: BCCH RSSI: low and high value range Range: -50 dBm to 34 dBm, Unit: dBm
- Bsic: enums.Bsic: NONVerified | VERified NONV: RSSI measurement without BSIC decoding VER: RSSI measurement with BSIC decoding

get(*cell*=<*Cell.Nr1: 1*>) → GetStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UEReport:NCELl:GSM:CELL<nr>
value: GetStruct = driver.sense.ueReport.ncell.gsm.cell.get(cell = repcap.Cell.
↪Nr1)
```

Returns the UE measurement report contents for GSM neighbor cell. See also 'Neighbor Cell Settings'.

param cell optional repeated capability selector. Default value: Nr1

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

7.3.2.1.2 Wcdma

class Wcdma

Wcdma 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.ueReport.ncell.wcdma.clone()
```

Subgroups

7.3.2.1.2.1 Cell

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UEReport:NCELl:WCDMa:CELL<Cell>
```

class Cell

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

class GetStruct

Response structure. Fields:

- Rscp: List[float]: CPICH RSCP: low and high value range Range: -120 dBm to -25 dBm , Unit: dBm
- Ecn_0: List[float]: CPICH Ec/No: low and high value range Range: -24 dB to 0 dB , Unit: dB
- Rssi: List[float]: CPICH RSSI Range: -50 dBm to 34 dBm, Unit: dBm
- Sfn_Cfn: List[float]: SFN-CFN time difference: low end high value range Range: 768 chips to 1280 chips , Unit: chips
- Pathloss: float: Range: 46 dB to 158 dB , Unit: dB

get(*cell*=<*Cell.Nr1: 1*>) → GetStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UEReport:NCELl:WCDMa:CELL<nr>
value: GetStruct = driver.sense.ueReport.ncell.wcdma.cell.get(cell = repcap.
↪Cell.Nr1)
```

Returns the UE measurement report contents for WCDMA neighbor cell. See also 'Neighbor Cells UTRA FDD'.

param cell optional repeated capability selector. Default value: Nr1

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

7.3.2.1.3 Lte

class Lte
Lte 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.ueReport.ncell.lte.clone()
```

Subgroups

7.3.2.1.3.1 Cell

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UEReport:NCELl:LTE:CELL<Cell>
```

class Cell
Cell commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class GetStruct
Response structure. Fields:

- Rsrp: List[float]: Range: -19.5 dB to -3 dB , Unit: dB
- Rsrq: List[float]: Range: -140 dBm to -44 dBm , Unit: dBm

get(*cell*=<*Cell.Nr1: 1*>) → GetStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UEReport:NCELl:LTE:CELL<nr>
value: GetStruct = driver.sense.ueReport.ncell.lte.cell.get(cell = repcap.Cell.
↪Nr1)
```

Returns the low and high value ranges reported for a selected LTE neighbor cell. See also 'Neighbor Cell Settings'.

param cell optional repeated capability selector. Default value: Nr1

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

7.3.3 UeCapability

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:HSUPa
SENSe:WCDMa:SIGNaling<Instance>:UECapability:HSDPa
SENSe:WCDMa:SIGNaling<Instance>:UECapability:GENeral
SENSe:WCDMa:SIGNaling<Instance>:UECapability:MRAT
SENSe:WCDMa:SIGNaling<Instance>:UECapability:MMODe
SENSe:WCDMa:SIGNaling<Instance>:UECapability:PUPLink
```

(continues on next page)

(continued from previous page)

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:PDOWnlink
SENSe:WCDMa:SIGNaling<Instance>:UECapability:RLC
SENSe:WCDMa:SIGNaling<Instance>:UECapability:PDCP
SENSe:WCDMa:SIGNaling<Instance>:UECapability:IMSVoice
```

class UeCapability

UeCapability commands group definition. 29 total commands, 4 Sub-groups, 10 group commands

class GeneralStruct

Structure for reading output parameters. Fields:

- Release: int: Access stratum release indicator, e.g. Rel. 99, Rel. 5 Range: 5 to 99
- Batt_Consum_Opt: enums.YesNoStatus: NO | YES Indicates whether the UE benefits from NW-based battery consumption optimization
- Mimo_Only_Single_Stream: enums.YesNoStatus: NO | YES Indicates whether the UE supports MIMO only single stream
- Emeas_Report: enums.YesNoStatus: NO | YES Indicates whether the UE supports E-UTRAN measurement reporting
- Adj_Frq_Mea_No_Cm: enums.YesNoStatus: NO | YES Indicates whether the UE supports adjacent frequency measurements without compressed mode
- In_Bfrq_Meas_No_Cm: enums.YesNoStatus: NO | YES Indicates whether the UE supports inter-band frequency measurements without compressed mode
- Sib_11_Bis: enums.YesNoStatus: NO | YES Indicates whether the UE supports system information block 11bis
- Csg: enums.YesNoStatus: NO | YES Indicates whether the UE supports closed subscriber group (CSG)
- Csg_Proximity: enums.YesNoStatus: NO | YES Indicates whether the UE supports CSG proximity indication
- Cell_Tx_Div_Dc: enums.YesNoStatus: NO | YES Indicates whether the UE supports cell-specific TX diversity in dual cell operation
- Ncell_Si_Acq: enums.YesNoStatus: NO | YES Indicates whether the UE supports a neighbor cell system information acquisition
- Cs_Vo_Hspa: enums.YesNoStatus: NO | YES Indicates whether the UE supports CS voice over HSPA
- Dc_Mimo_Diff_Bands: enums.YesNoStatus: NO | YES Indicates whether the UE supports dual cell with MIMO operation in different bands
- Utran_Anr: enums.YesNoStatus: NO | YES Indicates whether the UE supports ANR
- Um_Rlc_Re_Est_Re_Cnf: enums.YesNoStatus: NO | YES Indicates whether the UE supports UM RLC reestablishment via reconfiguration
- Rf_Mfbi: enums.YesNoStatus: NO | YES Indicates whether the UE supports multiple frequency band indicators
- Reserved: enums.YesNoStatus: NO | YES Reserved for future
- Ext_Meas: enums.YesNoStatus: NO | YES Indicates whether the UE supports extended measurements
- Fr_99_Prach: enums.YesNoStatus: NO | YES Indicates whether the UE supports fallback to R99 PRACH in CELL_FACH state and IDLE mode

- Conc_Deployment: enums.YesNoStatus: NO | YES Indicates whether the UE supports concurrent deployment of 2 ms and 10 ms TTI in a cell in CELL_FACH state and IDLE mode
- Tti_Align_Harq: enums.YesNoStatus: NO | YES Indicates whether the UE supports TTI alignment and per HARQ process activation and deactivation in CELL_FACH state and IDLE mode
- Mimodsr_4_X_4: enums.YesNoStatus: NO | YES Indicates whether the UE supports MIMO mode with four transmit antennas only with restriction to dual stream operation
- Ncmc_Mimo: enums.YesNoStatus: NO | YES Indicates whether the UE supports non-contiguous multi-cell operation on two, three or four cells with single gap in one band with MIMO
- Dsac_Ppac_Cell_Dch: enums.YesNoStatus: NO | YES Indicates whether the UE supports DSAC and PPAC update in CELL_DCH
- Acc_Grps_Acc_Ctrl: enums.YesNoStatus: NO | YES Indicates whether the UE supports access groups -based access control
- En_Tti_Switching: enums.YesNoStatus: NO | YES Indicates whether the UE supports enhanced TTI switching
- Impl_Grant: enums.YesNoStatus: NO | YES Indicates whether the UE supports implicit grant handling

class HsdpaStruct

Structure for reading output parameters. Fields:

- Hs_Pdsch: enums.YesNoStatus: NO | YES Indicates whether the UE supports the HS-PDSCH
- Dl_Cap_Hsd_Sch: int: Supported DPCH data rate in case an HS-DSCH is configured simultaneously Range: 32 kbit/s to 384 kbit/s, Unit: kbit/s
- Phys_Layer_Cat_R_5: int: HS-DSCH physical layer category of the UE for release 5 call setup Range: 1 to 24
- Phys_Layer_Cat_R_7: int: HS-DSCH physical layer category of the UE for release 7 call setup Range: 1 to 24
- Phys_Layer_Cat_R_8: int: HS-DSCH physical layer category of the UE for release 8 call setup Range: 1 to 24
- Phys_Layer_Cat_R_9: int: HS-DSCH physical layer category of the UE for release 9 call setup
- Hsdsch_Drx_Op: enums.YesNoStatus: NO | YES Indicates whether the UE supports the HS-DSCH DRX operation
- Hs_Scch_Less: enums.YesNoStatus: NO | YES Indicates whether the UE supports the HS-SCCH less operation
- Cell_Fach: enums.YesNoStatus: NO | YES Indicates whether the UE supports HS-PDSCH in CELL_FACH state
- Cell_Pc_Hurapch: enums.YesNoStatus: NO | YES Indicates whether the UE supports the HS-PDSCH in CELL_PCH and URA_PCH states
- Phys_Layer_Cat_R_10: int: HS-DSCH physical layer category of the UE for release 10 call setup Range: 29 to 32
- Ma_Cehs: enums.YesNoStatus: NO | YES Indicates whether the UE supports the MAC-ehs
- Phys_Layer_Cat_R_11: int: HS-DSCH physical layer category of the UE for release 11 call setup
- Hsdpcch_Poff_Ext: enums.YesNoStatus: NO | YES Indicates whether the UE supports the values 9 and 10 of deltaACK, deltaNACK and deltaCQI power offset

- Drx_Op_2nd_C: enums.YesNoStatus: NO | YES Indicates whether the UE supports HS-DSCH DRX operation with second DRX cycle in CELL_FACH state
- Nb_Trg_Hsdpcch: enums.YesNoStatus: NO | YES Indicates whether the UE supports NodeB triggered HS-DPCCH transmission in CELL_FACH state
- Hsdpcch_Ov_Hd_Rd: enums.YesNoStatus: NO | YES Indicates whether the UE supports HS-DPCCH overhead reduction

class HsupaStruct

Structure for reading output parameters. Fields:

- Hsupa: enums.YesNoStatus: NO | YES Indicates whether the UE supports HSUPA
- Phys_Layer_Cat_R_6: int: E-DCH physical layer category of the UE for release 6 call setup Range: 1 to 6
- Phys_Layer_Cat_R_9: int: E-DCH physical layer category of the UE for release 9 call setup Range: 8 to 9
- Phys_Layer_Cat_R_7: int: E-DCH physical layer category of the UE for release 7 call setup Range: 7 to 7
- Phys_Layer_Cat_R_11: int: E-DCH physical layer category of the UE for release 11 call setup
- Ergch_Bic_Trl: enums.YesNoStatus: NO | YES Indicates whether the UE supports common E-RGCH-based interference control in CELL_FACH state

class ImsVoiceStruct

Structure for reading output parameters. Fields:

- Vo_Utra_Ps_Hs: enums.YesNoStatus: NO | YES Indicates if a UE supports IMS voice over UTRA PS HSPA connections
- Srvcc_Utra_Utra: enums.YesNoStatus: NO | YES Indicates if a UE supports the single radio voice call continuity (SRVCC) from UTRA PS HS to UTRA CS
- Srvcc_Utra_Geran: enums.YesNoStatus: NO | YES Indicates if a UE supports SRVCC from UTRA PS HS to GERAN CS
- Rs_Rvcc_Ucs_Eu_Fdd: enums.YesNoStatus: NO | YES Indicates whether the UE supports reverse single radio voice call continuity (rSRVCC) handover from UTRA CS to EUTRA FDD
- Rs_Rvcc_Ucs_Eu_Tdd: enums.YesNoStatus: NO | YES Indicates whether the UE supports rSRVCC handover from UTRA CS to EUTRA TDD

class MratStruct

Structure for reading output parameters. Fields:

- Support_Gsm: enums.YesNoStatus: NO | YES Indicates whether the UE supports GSM
- Multi_Carrier: enums.YesNoStatus: NO | YES Indicates whether the UE supports multi-carrier mode
- Utran_Geran: enums.YesNoStatus: NO | YES Indicates whether the UE supports UTRAN to GERAN NACC
- Handover_Gan: enums.YesNoStatus: NO | YES Indicates whether the UE supports CS handover to GAN
- Ps_Inter_Rat: enums.YesNoStatus: NO | YES Indicates whether the UE supports Inter-RAT PS handover
- Cipher_Alg_Uea_0: enums.YesNoStatus: NO | YES Indicates whether the UE supports ciphering algorithm UEA0

- Cipher_Alg_Uea_1: enums.YesNoStatus: NO | YES Indicates whether the UE supports ciphering algorithm UEA1
- Integrity_Uia_1: enums.YesNoStatus: NO | YES Indicates whether the UE supports integrity algorithm UIA1
- Cipher_Alg_Uea_2: enums.YesNoStatus: NO | YES Indicates whether the UE supports ciphering algorithm UEA2
- Integrity_Uia_2: enums.YesNoStatus: NO | YES Indicates whether the UE supports integrity algorithm UIA2
- Trgt_Cell_Pre_Cfg: enums.YesNoStatus: NO | YES Indicates whether the UE supports target cell preconfiguration
- Ps_Handover_Gan: enums.YesNoStatus: NO | YES Indicates whether the UE supports PS handover to GAN
- Eutra_Fdd: enums.YesNoStatus: NO | YES Indicates whether the UE supports E-UTRA FDD
- Eutra_Inter_Rat: enums.YesNoStatus: NO | YES Indicates whether the UE supports inter-RAT E-UTRA handover
- U_2_Eu_Tra_Rrci_Dle: enums.YesNoStatus: NO | YES Indicates whether the UE supports cell reselection from UTRA CELL_PCH or URA_PCH to E-UTRA RRC_IDLE
- Prio_Res_Utran: enums.YesNoStatus: NO | YES Indicates whether the UE supports priority reselection in UTRAN
- Eutra_Fdd_Cfch: enums.YesNoStatus: NO | YES Indicates whether the UE supports E-UTRA measurements and reporting for CELL_FACH for E-UTRA FDD
- Eutra_Tdd_Cdch: enums.YesNoStatus: NO | YES Indicates whether the UE supports E-UTRA measurements and reporting for CELL_FACH for E-UTRA TDD
- Eutra_Mfbi: enums.YesNoStatus: NO | YES Indicates whether the UE supports E-UTRA multiple frequency band indicator measurements
- Wlan_Ran_Rules: enums.YesNoStatus: NO | YES Indicates whether the UE supports RAN-assisted WLAN interworking RAN rules
- Wlan_And_Sf: enums.YesNoStatus: NO | YES Indicates whether the UE supports RAN-assisted WLAN interworking ANDSF policies

`class PdcpStruct`

Structure for reading output parameters. Fields:

- Srns: enums.YesNoStatus: NO | YES Support of lossless SRNS relocation
- Rfc_2507: enums.YesNoStatus: NO | YES Support of IP header compression according to RFC 2507
- Rfc_3095: enums.YesNoStatus: NO | YES Support of robust header compression according to RFC 3095
- Rfc_3095_Ctx_Reloc: enums.YesNoStatus: NO | YES Support of context relocation applied to the RFC 3095 header compression protocol
- Header_Comp: int: Maximum header compression context size supported by the UE. This parameter is only applicable if the UE supports header compression according to RFC 2507 Range: 1024 to 131072
- Max_Rohc: int: Maximum number of header compression context sessions supported by the UE. This parameter is only applicable if the UE supports header compression according to RFC3095. Range: 2 to 16384

- Reverse_Decomp: int: Number of packets that can be reverse decompressed by the decompressor in the UE Range: 0 to 65535
- Pdu_Size_Change: enums.YesNoStatus: NO | YES Support of lossless DL RLC PDU size change
- Rfc_3095_Rspace: int: 16384 | 32768 | 65536 | 131072 Support of RFC 3095 relocation space Unit: byte

class PdownlinkStruct

Structure for reading output parameters. Fields:

- Simult_Transp_Ch: int: Maximum number of downlink transport channels that the UE is capable to process simultaneously, not considering the rate of each transport channel Range: 4 to 32
- Simult_Cc_Tr_Ch: int: Maximum number of downlink coded composite transport channels (CCTrCH) that the UE is capable to process simultaneously. Interpret CCTrCH as consisting of DCH, FACH or DSCH. Range: 1 to 8
- Tti_Transp_Block: int: Maximum total number of transport blocks received within transmission time intervals (TTIs) that end within the same 10 ms interval. This value includes all transport blocks that are to be simultaneously received by the UE on DCH, FACH, PCH and DSCH transport channels. Range: 4 to 512
- Number_Of_Tfc: int: Maximum number of transport format combinations (TFC) in a downlink transport format combination set that the UE can store Range: 16 to 1024
- Number_Of_Tf: int: Maximum number of downlink transport formats (TF) that the UE can store, where all transport formats for all downlink transport channels are counted Range: 32 to 1024
- Turbo_Decoding: enums.YesNoStatus: NO | YES Support of turbo decoding
- Rx_Bits_All: int: Maximum number of bits of all transport blocks being received at an arbitrary time instant. All bits are considered. Range: 640 bits to 163840 bits, Unit: bits
- Rx_Bits_Conv: int: Maximum number of bits of all transport blocks being received at an arbitrary time instant. Only convolutionally coded bits are considered. Range: 640 bits to 163840 bits, Unit: bits
- Rx_Bits_Turbo: int: Maximum number of bits of all transport blocks being received at an arbitrary time instant. Only turbo coded bits are considered. Range: 640 bits to 163840 bits, Unit: bits
- Dpcch_Codes: int: Maximum number of DPCH codes to be simultaneously received. For DPCH in soft/softer handover, each DPCH is only calculated once. The capability does not include codes used for S-CCPCH. Range: 1 to 8
- Physical_Ch_Bits: int: Maximum number of physical channel bits received in any 10 ms interval (DPCH, PDSCH, S-CCPCH) . For DPCH in soft/softer handover, each DPCH is only calculated once. Range: 600 bits to 76800 bits, Unit: bits
- Sf_512: enums.YesNoStatus: NO | YES Support of spreading factor (SF) 512 in downlink.
- Ma_Ciis: enums.YesNoStatus: NO | YES Support of MAC-i/is entity handling E-DCH
- Fdpch: enums.YesNoStatus: NO | YES Support of FDD physical channel F-DPCH
- Enhanced_Fdpch: enums.YesNoStatus: NO | YES Support of FDD physical channel enhanced F-DPCH
- Dc_Henh: enums.DchEnhanced: NO | BASic | FULL Support of DCH enhancements
- Simult_Dch_Enh_Cm: enums.YesNoStatus: NO | YES Support of simultaneous DCH enhancements and CM

- Sdch_Enh_Dpcch: enums.YesNoStatus: NO | YES Support of simultaneous DCH enhancements and DPCCH DTX
- Drx_Enh: enums.YesNoStatus: NO | YES Support of DRX enhancements
- Dpcch_2_Trx: enums.YesNoStatus: NO | YES Support of FDD DPCCH2 transmission
- Ftpi_Ch_Feedback: enums.YesNoStatus: NO | YES Support of FDD F-TPICH feedback from the multiflow assisting cell

class PupLinkStruct

Structure for reading output parameters. Fields:

- Simult_Transp_Ch: int: Maximum number of uplink transport channels that the UE is capable to process simultaneously, not considering the rate of each transport channel Range: 4 to 32
- Simult_Cc_Tr_Ch: int: Maximum number of uplink coded composite transport channels (CCTrCH) that the UE is capable to process simultaneously Range: 1 to 8
- Tti_Transp_Block: int: Maximum total number of transport blocks transmitted within transmission time intervals (TTI) that start at the same time Range: 4 to 512
- Number_Of_Tfc: int: Maximum number of transport format combinations (TFC) in an uplink transport format combination set that the UE can store Range: 16 to 1024
- Number_Of_Tf: int: Maximum number of uplink transport formats (TF) that the UE can store, where all transport formats for all uplink transport channels are counted Range: 32 to 1024
- Turbo_Decoding: enums.YesNoStatus: NO | YES Support of turbo decoding
- Tx_Bits_All: int: Maximum number of bits of all transport blocks being transmitted at an arbitrary time instant. All bits are considered. Range: 640 bits to 163840 bits, Unit: bits
- Tx_Bits_Conv: int: Maximum number of bits of all transport blocks being transmitted at an arbitrary time instant. Only convolutionally coded bits are considered. Range: 640 bits to 163840 bits, Unit: bits
- Tx_Bits_Turbo: int: Maximum number of bits of all transport blocks being transmitted at an arbitrary time instant. Only turbo coded bits are considered. Range: 640 bits to 163840 bits, Unit: bits
- Dpd_Ch_Bits: int: Maximum number of DPDCH bits the UE can transmit in 10 ms. The value applies to UE operation in non-compressed mode (if the value is 9600) or in both compressed and non-compressed mode (if the value is 9600) . Range: 600 bits to 57600 bits, Unit: bits
- Dpcch_Dtx: enums.YesNoStatus: NO | YES Support of discontinuous uplink DPCCH transmission
- Slot_Format_4: enums.YesNoStatus: NO | YES Support of DPCCH slot format 4
- Common_Edch: enums.YesNoStatus: NO | YES Support of common E-DCH
- Edpcch_Pwr_Boost: enums.YesNoStatus: NO | YES Support of E-DPCCH power boosting
- Edpdch_Pwr_Interpol: enums.YesNoStatus: NO | YES Support of E-DPCCH power interpolation
- Dtx_Enh: enums.YesNoStatus: NO | YES Support of DTX enhancements
- Srv_Edc_Hcd_Op: enums.YesNoStatus: NO | YES Support of FDD serving E-DCH cell decoupling operation
- Rlwo_Fdpch: enums.YesNoStatus: NO | YES Support of FDD radio link without DPCH or F-DPCH

class RlcStruct

Structure for reading output parameters. Fields:

- Am_Buffer_Size: int: Maximum total buffer size across all RLC AM entities supported by the UE Range: 10 to 1000

- Max_Rlc_Window: int: Maximum RLC window size supported by the UE Range: 0 to 4095
- Am_Entities: int: Maximum number of AM entities supported by the UE Range: 3 to 30
- Two_Logical_Ch: enums.YesNoStatus: NO | YES Support of AM entity configurated with two logical channels

get_general() → GeneralStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:GENeral
value: GeneralStruct = driver.sense.ueCapability.get_general()
```

Returns general UE capability information.

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

get_hsdpa() → HsdpaStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:HSDPa
value: HsdpaStruct = driver.sense.ueCapability.get_hsdpa()
```

Returns UE capability information related to HSDPA.

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

get_hsupa() → HsupaStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:HSUPa
value: HsupaStruct = driver.sense.ueCapability.get_hsupa()
```

Returns UE capability information related to HSUPA.

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

get_ims_voice() → ImsVoiceStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:IMSVoice
value: ImsVoiceStruct = driver.sense.ueCapability.get_ims_voice()
```

Indicates the IMS voice capability of the UE as defined in 3GPP TS 25.331, section 10.3.3.14b.

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

get_mmode() → RsCmwWcdmaSig.enums.UtraMode

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:MMODe
value: enums.UtraMode = driver.sense.ueCapability.get_mmode()
```

Returns UE capability information indicating whether the UE supports UTRA FDD or TDD or both.

return utra: FDD | TDD | BOTH

get_mrat() → MratStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:MRAT
value: MratStruct = driver.sense.ueCapability.get_mrat()
```

Returns UE capability information indicating the radio access technologies (RAT) that the UE supports.

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

get_pdcp() → PdcpStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:PDCP
value: PdcpStruct = driver.sense.ueCapability.get_pdcp()
```

Returns UE capability information indicating in which way the UE supports the packet data convergence protocol (PDCP) described in 3GPP TS 25.323

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

get_pdownlink() → PdownlinkStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:PDOWnlink
value: PdownlinkStruct = driver.sense.ueCapability.get_pdownlink()
```

Returns UE capability information describing the capacity of the UE to process and store downlink channels.

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

get_pup_link() → PupLinkStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:PUPLink
value: PupLinkStruct = driver.sense.ueCapability.get_pup_link()
```

Returns UE capability information describing the capacity of the UE to process and store uplink channels.

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

get_rlc() → RlcStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:RLC
value: RlcStruct = driver.sense.ueCapability.get_rlc()
```

Returns UE capability information indicating in which way the UE supports the radio link control acknowledged mode (RLC AM) .

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

Cloning the Group

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

Subgroups

7.3.3.1 Codec

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:CODec:GSM
SENSe:WCDMa:SIGNaling<Instance>:UECapability:CODec:UMTS
```

class Codec

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

get_gsm() → List[RsCmwWcdmaSig.enums.YesNoStatus]

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:CODec:GSM
value: List[enums.YesNoStatus] = driver.sense.ueCapability.codec.get_gsm()
```

Indicates codec list supported by the UE in GSM and UMTS networks. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

return supported: NO | YES 14 values indicate support for: 1: GSM FR 2: GSM HR 3: GSM EFR 4: FR AMR 5: HR AMR 6: UMTS AMR 7: UMTS AMR 2 8: TDMA EFR 9: PDC EFR 10: FR AMR-WB 11: UMTS AMR-WB 12: OHR AMR 13: OFR AMR-WB 14: OHR AMR-WB

get_umts() → List[RsCmwWcdmaSig.enums.YesNoStatus]

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:CODec:UMTS
value: List[enums.YesNoStatus] = driver.sense.ueCapability.codec.get_umts()
```

Indicates codec list supported by the UE in GSM and UMTS networks. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

return supported: NO | YES 14 values indicate support for: 1: GSM FR 2: GSM HR 3: GSM EFR 4: FR AMR 5: HR AMR 6: UMTS AMR 7: UMTS AMR 2 8: TDMA EFR 9: PDC EFR 10: FR AMR-WB 11: UMTS AMR-WB 12: OHR AMR 13: OFR AMR-WB 14: OHR AMR-WB

7.3.3.2 Measurement

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:MEASurement
```

class Measurement

Measurement commands group definition. 5 total commands, 1 Sub-groups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Inter_Freq_Detect: enums.YesNoStatus: NO | YES Indicates whether the UE is able to measure inter-frequency detected set.
- Enh_Inter_Freq: enums.YesNoStatus: NO | YES Indicates whether the UE requires compressed mode for measurements on two additional frequencies.
- Freq_Specific_Cm: enums.YesNoStatus: NO | YES Indicates whether the UE can apply compressed mode outside of the used frequency bands only to the configured frequencies. This information is relevant only for the dual band operation.
- Intr_Frq_Cc_Wo_Cm: enums.YesNoStatus: NO | YES Indicates whether the UE requires compressed mode to measure on the frequencies which are configured for HS-DSCH operation and associated with the secondary serving HS-DSCH cells
- Ceds_Meas: enums.YesNoStatus: NO | YES Indicates whether the UE supports exclusion of cells from intra-frequency detected set measurements
- Wr_Srq_Fdd_Meas: enums.YesNoStatus: NO | YES Indicates whether the UE is able to perform wideband RSRQ FDD measurements
- Ev_2_Grep_Sec_Dl_Frq: enums.YesNoStatus: NO | YES Indicates whether the UE supports event 2G reporting on a secondary DL frequency
- Ext_Rs_Rq_Lwr_Rng: enums.YesNoStatus: NO | YES Indicates whether the UE supports extended RSRQ lower value range
- Rsrq_On_All_Sym: enums.YesNoStatus: NO | YES Indicates whether the UE supports RSRQ on all symbols
- Inc_Ue_Cr_Mn_Utra: enums.YesNoStatus: NO | YES Indicates whether the UE supports increased number of UTRA carrier monitoring in connected and idle mode
- Inc_Ue_Cr_Mn_Eutra: enums.YesNoStatus: NO | YES Indicates whether the UE supports increased number of E-UTRA carrier monitoring in connected and idle mode
- Enh_Uph_Reporting: enums.YesNoStatus: NO | YES Indicates whether the UE supports enhanced UPH reporting
- Escce_1_Cop: enums.YesNoStatus: NO | YES Indicates whether the UE supports enhanced serving cell change for event 1c operation
- Cri_Reporting: enums.YesNoStatus: NO | YES Indicates whether the UE supports cell resellection indication reporting

get_value() → ValueStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:MEASurement
value: ValueStruct = driver.sense.ueCapability.measurement.get_value()
```

Queries the UE capabilities related to inter-frequency measurements.

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.sense.ueCapability.measurement.clone()
```

Subgroups

7.3.3.2.1 Cmode

class Cmode

Cmode 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.sense.ueCapability.measurement.cmode.clone()
```

Subgroups

7.3.3.2.1.1 Wcdma

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:MEASurement:CMODe:WCDMa
```

class Wcdma

Wcdma commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get(*band: RsCmwWcdmaSig.enums.CompressedModeBand*) → List[RsCmwWcdmaSig.enums.CompressedMode]

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:MEASurement:CMODe:WCDMa
value: List[enums.CompressedMode] = driver.sense.ueCapability.measurement.cmode.
↪wcdma.get(band = enums.CompressedModeBand.OB1)
```

Returns the UE capabilities for WCDMA and WCDMA multicarrier neighbor cell measurements-related compressed mode.

param band OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16 | OB17 | OB18 | OB19 | OB20 | OB21 | OB22 | OB25 | OB26 | OB32 OB1, ..., OB22: WCDMA operating band I to XXII OB25, OB26, OB32: WCDMA operating band XXV, XXVI and XXXII

return compressed_mode: NN | NY | YN | YY NN: compressed mode for the neighbor cell measurement not required (UL and DL) NY: compressed mode for the neighbor cell measurement required in DL only YN: compressed mode for the neighbor cell

measurement required in UL only YY: compressed mode for the neighbor cell measurement required in UL and DL

Cloning the Group

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

Subgroups

7.3.3.2.1.2 Mcarrier

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:MEASurement:CMODe:WCDMa:MCARrier
```

class Mcarrier

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

get(*band: RsCmwWcdmaSig.enums.CompressedModeBand*) → RsCmwWcdmaSig.enums.CompressedMode

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>
↪:UECapability:MEASurement:CMODe:WCDMa:MCARrier
value: enums.CompressedMode = driver.sense.ueCapability.measurement.cmode.wcdma.
↪mcarrier.get(band = enums.CompressedModeBand.OB1)
```

Returns the UE capabilities for WCDMA and WCDMA multicarrier neighbor cell measurements-related compressed mode.

param band OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16 | OB17 | OB18 | OB19 | OB20 | OB21 | OB22 | OB25 | OB26 | OB32 OB1, ..., OB22: WCDMA operating band I to XXII OB25, OB26, OB32: WCDMA operating band XXV, XXVI and XXXII

return compressed_mode: NN | NY | YN | YY NN: compressed mode for the neighbor cell measurement not required (UL and DL) NY: compressed mode for the neighbor cell measurement required in DL only YN: compressed mode for the neighbor cell measurement required in UL only YY: compressed mode for the neighbor cell measurement required in UL and DL

7.3.3.2.1.3 Gsm

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:MEASurement:CMODe:GSM
```

class Gsm

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

get(*band: RsCmwWcdmaSig.enums.CompressedModeBand*) → List[RsCmwWcdmaSig.enums.CompressedMode]

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:MEASurement:CMODe:GSM
value: List[enums.CompressedMode] = driver.sense.ueCapability.measurement.cmode.
↪gsm.get(band = enums.CompressedModeBand.OB1)
```

Returns the UE capabilities for GSM neighbor cell measurements-related compressed mode.

param band OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16 | OB17 | OB18 | OB19 | OB20 | OB21 | OB22 | OB25 | OB26 | OB32 OB1, ..., OB22: WCDMA operating band I to XXII OB25, OB26, OB32: WCDMA operating band XXV, XXVI and XXXII

return compressed_mode: NN | NY | YN | YY NN: compressed mode for the neighbor cell measurement not required (UL and DL) NY: compressed mode for the neighbor cell measurement required in DL only YN: compressed mode for the neighbor cell measurement required in UL only YY: compressed mode for the neighbor cell measurement required in UL and DL

7.3.3.2.1.4 Lte

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:MEASurement:CMODe:LTE
```

class Lte

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

get(*band: RsCmwWcdmaSig.enums.CompressedModeBand*) → List[RsCmwWcdmaSig.enums.CompressedMode]

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:MEASurement:CMODe:LTE
value: List[enums.CompressedMode] = driver.sense.ueCapability.measurement.cmode.
↪lte.get(band = enums.CompressedModeBand.OB1)
```

Returns the UE capabilities for LTE neighbor cell measurements-related compressed mode.

param band OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16 | OB17 | OB18 | OB19 | OB20 | OB21 | OB22 | OB25 | OB26 | OB32 OB1, ..., OB22: WCDMA operating band I to XXII OB25, OB26, OB32: WCDMA operating band XXV, XXVI and XXXII

return compressed_mode: NN | NY | YN | YY NN: compressed mode for the neighbor cell measurement not required (UL and DL) NY: compressed mode for the neighbor cell measurement required in DL only YN: compressed mode for the neighbor cell measurement required in UL only YY: compressed mode for the neighbor cell measurement required in UL and DL

7.3.3.3 UePosition

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:UEPosition
```

class UePosition

UePosition commands group definition. 7 total commands, 1 Sub-groups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Location_Method: enums.YesNoStatus: NO | YES Indicates if a UE can measure its location by some means unrelated to UTRAN (e.g. if the UE has access to a standalone GPS receiver)
- Network_Agps: enums.NetworkAndGps: NONE | NETWork | UE | BOTH Indicates if a UE supports the assisted GPS schemes network-based and/or UE-based
- Ref_Time_Gps: enums.YesNoStatus: NO | YES Indicates UE capability to measure GPS reference time as defined in 3GPP TS 25.215
- Ipdl: enums.YesNoStatus: NO | YES Indicates UE capability to use idle periods in the downlink (IPDL) to enhance its 'SFN-SFN observed time difference – type 2' measurement
- Otdoa: enums.YesNoStatus: NO | YES Indicates if a UE supports the observed time difference of arrival (OTDOA) UE-based schemes
- Rx_Tx_Time_Diff: enums.YesNoStatus: NO | YES Indicates UE capability to measure the Rx-Tx time difference type 2
- Cell_Ur_Apch: enums.YesNoStatus: NO | YES Indicates whether the UE positioning measurements using the assisted GPS method are valid in CELL_PCH and URA_PCH RRC states
- Sfn_Sfn_Time_Diff: enums.YesNoStatus: NO | YES Indicates UE capability to perform the SFN-SFN observed time difference type 2 measurement

get_value() → ValueStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:UEPosition
value: ValueStruct = driver.sense.ueCapability.uePosition.get_value()
```

Returns UE capability information related to UE positioning.

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.sense.ueCapability.uePosition.clone()
```

Subgroups

7.3.3.3.1 Ganss

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:UEPosition:GANSs:GALileo
SENSe:WCDMa:SIGNaling<Instance>:UECapability:UEPosition:GANSs:SBAS
SENSe:WCDMa:SIGNaling<Instance>:UECapability:UEPosition:GANSs:MGPS
SENSe:WCDMa:SIGNaling<Instance>:UECapability:UEPosition:GANSs:QZSS
SENSe:WCDMa:SIGNaling<Instance>:UECapability:UEPosition:GANSs:GLONass
SENSe:WCDMa:SIGNaling<Instance>:UECapability:UEPosition:GANSs
```

class Ganss

Ganss commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

class GalileoStruct

Structure for reading output parameters. Fields:

- Supported: enums.YesNoStatus: NO | YES Indicates if a UE supports the navigation standard indicated by the last mnemonic
- Mode: enums.UeNaviSupport: NONE | NETWork | UE | NUE Indicates if a UE supports the 'network-based' and/or 'UE-based' navigation standard indicated by the last mnemonic
- Signal_Id: int: The GANSS signal ID encodes the identification of the signal for each GANSS. It depends on the GANSS ID as specified in 3GPP TS 25.331, section 10.3.3.45a.
- Signal_Ids_Ext: int: GANSS signal IDs extension specifies the UE capability to measure on more than one GANSS signal and which signals are supported (see 3GPP TS 25.331, section 10.3.3.45, note 2) .
- Timing_Cell_Frms: enums.YesNoStatus: NO | YES Support of GANSS timing of cell frames measurement
- Carrier_Phase: enums.YesNoStatus: NO | YES Support of GANSS carrier-phase measurement
- Non_Native_Assist: enums.YesNoStatus: NO | YES Support of non-native assistance choices
- Sbas_Id: int: Coding is specified in 3GPP TS 25.331, section 10.3.3.45, note 1. This parameter is only available for SBAS standard.

class GlonassStruct

Structure for reading output parameters. Fields:

- Supported: enums.YesNoStatus: NO | YES Indicates if a UE supports the navigation standard indicated by the last mnemonic
- Mode: enums.UeNaviSupport: NONE | NETWork | UE | NUE Indicates if a UE supports the 'network-based' and/or 'UE-based' navigation standard indicated by the last mnemonic
- Signal_Id: int: The GANSS signal ID encodes the identification of the signal for each GANSS. It depends on the GANSS ID as specified in 3GPP TS 25.331, section 10.3.3.45a.
- Signal_Ids_Ext: int: GANSS signal IDs extension specifies the UE capability to measure on more than one GANSS signal and which signals are supported (see 3GPP TS 25.331, section 10.3.3.45, note 2) .
- Timing_Cell_Frms: enums.YesNoStatus: NO | YES Support of GANSS timing of cell frames measurement
- Carrier_Phase: enums.YesNoStatus: NO | YES Support of GANSS carrier-phase measurement
- Non_Native_Assist: enums.YesNoStatus: NO | YES Support of non-native assistance choices

- Sbas_Id: int: Coding is specified in 3GPP TS 25.331, section 10.3.3.45, note 1. This parameter is only available for SBAS standard.

class MgpsStruct

Structure for reading output parameters. Fields:

- Supported: enums.YesNoStatus: NO | YES Indicates if a UE supports the navigation standard indicated by the last mnemonic
- Mode: enums.UeNaviSupport: NONE | NETWork | UE | NUE Indicates if a UE supports the 'network-based' and/or 'UE-based' navigation standard indicated by the last mnemonic
- Signal_Id: int: The GANSS signal ID encodes the identification of the signal for each GANSS. It depends on the GANSS ID as specified in 3GPP TS 25.331, section 10.3.3.45a.
- Signal_Ids_Ext: int: GANSS signal IDs extension specifies the UE capability to measure on more than one GANSS signal and which signals are supported (see 3GPP TS 25.331, section 10.3.3.45, note 2) .
- Timing_Cell_Frms: enums.YesNoStatus: NO | YES Support of GANSS timing of cell frames measurement
- Carrier_Phase: enums.YesNoStatus: NO | YES Support of GANSS carrier-phase measurement
- Non_Native_Assist: enums.YesNoStatus: NO | YES Support of non-native assistance choices
- Sbas_Id: int: Coding is specified in 3GPP TS 25.331, section 10.3.3.45, note 1. This parameter is only available for SBAS standard.

class QzssStruct

Structure for reading output parameters. Fields:

- Supported: enums.YesNoStatus: NO | YES Indicates if a UE supports the navigation standard indicated by the last mnemonic
- Mode: enums.UeNaviSupport: NONE | NETWork | UE | NUE Indicates if a UE supports the 'network-based' and/or 'UE-based' navigation standard indicated by the last mnemonic
- Signal_Id: int: The GANSS signal ID encodes the identification of the signal for each GANSS. It depends on the GANSS ID as specified in 3GPP TS 25.331, section 10.3.3.45a.
- Signal_Ids_Ext: int: GANSS signal IDs extension specifies the UE capability to measure on more than one GANSS signal and which signals are supported (see 3GPP TS 25.331, section 10.3.3.45, note 2) .
- Timing_Cell_Frms: enums.YesNoStatus: NO | YES Support of GANSS timing of cell frames measurement
- Carrier_Phase: enums.YesNoStatus: NO | YES Support of GANSS carrier-phase measurement
- Non_Native_Assist: enums.YesNoStatus: NO | YES Support of non-native assistance choices
- Sbas_Id: int: Coding is specified in 3GPP TS 25.331, section 10.3.3.45, note 1. This parameter is only available for SBAS standard.

class SbasStruct

Structure for reading output parameters. Fields:

- Supported: enums.YesNoStatus: NO | YES Indicates if a UE supports the navigation standard indicated by the last mnemonic
- Mode: enums.UeNaviSupport: NONE | NETWork | UE | NUE Indicates if a UE supports the 'network-based' and/or 'UE-based' navigation standard indicated by the last mnemonic
- Signal_Id: int: The GANSS signal ID encodes the identification of the signal for each GANSS. It depends on the GANSS ID as specified in 3GPP TS 25.331, section 10.3.3.45a.

- Signal_Ids_Ext: int: GANSS signal IDs extension specifies the UE capability to measure on more than one GANSS signal and which signals are supported (see 3GPP TS 25.331, section 10.3.3.45, note 2) .
- Timing_Cell_Frms: enums.YesNoStatus: NO | YES Support of GANSS timing of cell frames measurement
- Carrier_Phase: enums.YesNoStatus: NO | YES Support of GANSS carrier-phase measurement
- Non_Native_Assist: enums.YesNoStatus: NO | YES Support of non-native assistance choices
- Sbas_Id: int: Coding is specified in 3GPP TS 25.331, section 10.3.3.45, note 1. This parameter is only available for SBAS standard.

class ValueStruct

Structure for reading output parameters. Fields:

- Galileo: enums.YesNoStatus: NO | YES Indicates if a UE supports Galileo standard
- Sbas: enums.YesNoStatus: NO | YES Indicates if a UE supports the satellite-based augmentation system
- Modernized_Gps: enums.YesNoStatus: NO | YES Indicates if a UE supports the modernized global positioning system
- Qzss: enums.YesNoStatus: NO | YES Indicates if a UE supports the quasi-zenith satellite system
- Glonass: enums.YesNoStatus: NO | YES Indicates if a UE supports the global navigation satellite system

get_galileo() → GalileoStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:UEPosition:GANSs:GALileo
value: GalileoStruct = driver.sense.ueCapability.uePosition.ganss.get_galileo()
```

Returns UE capability information related to the navigation standards indicated by the last mnemonic: Galileo, global navigation satellite system (GLONASS) , modernized global positioning system (GPS) , quasi-zenith satellite system (QZSS) , satellite-based augmentation system (SBAS)

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

get_glonass() → GlonassStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:UEPosition:GANSs:GLONass
value: GlonassStruct = driver.sense.ueCapability.uePosition.ganss.get_glonass()
```

Returns UE capability information related to the navigation standards indicated by the last mnemonic: Galileo, global navigation satellite system (GLONASS) , modernized global positioning system (GPS) , quasi-zenith satellite system (QZSS) , satellite-based augmentation system (SBAS)

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

get_mgps() → MgpsStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:UEPosition:GANSs:MGPS
value: MgpsStruct = driver.sense.ueCapability.uePosition.ganss.get_mgps()
```

Returns UE capability information related to the navigation standards indicated by the last mnemonic: Galileo, global navigation satellite system (GLONASS) , modernized global positioning system (GPS) , quasi-zenith satellite system (QZSS) , satellite-based augmentation system (SBAS)

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

get_qzss() → QzssStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:UEPosition:GANSs:QZSS
value: QzssStruct = driver.sense.ueCapability.uePosition.ganss.get_qzss()
```

Returns UE capability information related to the navigation standards indicated by the last mnemonic: Galileo, global navigation satellite system (GLONASS) , modernized global positioning system (GPS) , quasi-zenith satellite system (QZSS) , satellite-based augmentation system (SBAS)

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

get_sbas() → SbasStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:UEPosition:GANSs:SBAS
value: SbasStruct = driver.sense.ueCapability.uePosition.ganss.get_sbas()
```

Returns UE capability information related to the navigation standards indicated by the last mnemonic: Galileo, global navigation satellite system (GLONASS) , modernized global positioning system (GPS) , quasi-zenith satellite system (QZSS) , satellite-based augmentation system (SBAS)

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

get_value() → ValueStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:UEPosition:GANSs
value: ValueStruct = driver.sense.ueCapability.uePosition.ganss.get_value()
```

Returns UE capability information related to the Galileo and additional navigation satellite systems (GANSS) .

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

7.3.3.4 RfParameter

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:RFParameter:BCList
SENSe:WCDMa:SIGNaling<Instance>:UECapability:RFParameter
```

class RfParameter

RfParameter commands group definition. 5 total commands, 2 Sub-groups, 2 group commands

class BcListStruct

Structure for reading output parameters. Fields:

- Bcomb_1: enums.YesNoStatus: NO | YES Indicates if the UE supports the band combination 1+8
- Bcomb_2: enums.YesNoStatus: NO | YES Indicates if the UE supports the band combination 2+4
- Bcomb_3: enums.YesNoStatus: NO | YES Indicates if the UE supports the band combination 1+5
- Bcomb_4: enums.YesNoStatus: NO | YES Indicates if the UE supports the band combination 1+6
- Bcomb_5: enums.YesNoStatus: NO | YES Indicates if the UE supports the band combination 2+5

class ValueStruct

Structure for reading output parameters. Fields:

- Band_Supported_1: enums.YesNoStatus: No parameter help available
- Power_Class_1: int: UE power class for band I Range: 1 to 4
- Band_Supported_2: enums.YesNoStatus: No parameter help available
- Power_Class_2: int: No parameter help available
- Band_Supported_3: enums.YesNoStatus: No parameter help available
- Power_Class_3: int: No parameter help available
- Band_Supported_4: enums.YesNoStatus: No parameter help available
- Power_Class_4: int: No parameter help available
- Band_Supported_5: enums.YesNoStatus: No parameter help available
- Power_Class_5: int: No parameter help available
- Band_Supported_6: enums.YesNoStatus: No parameter help available
- Power_Class_6: int: No parameter help available
- Band_Supported_7: enums.YesNoStatus: No parameter help available
- Power_Class_7: int: No parameter help available
- Band_Supported_8: enums.YesNoStatus: No parameter help available
- Power_Class_8: int: No parameter help available
- Band_Supported_9: enums.YesNoStatus: No parameter help available
- Power_Class_9: int: No parameter help available
- Band_Supported_10: enums.YesNoStatus: No parameter help available
- Power_Class_10: int: No parameter help available
- Band_Supported_11: enums.YesNoStatus: No parameter help available
- Power_Class_11: int: No parameter help available
- Band_Supported_12: enums.YesNoStatus: No parameter help available
- Power_Class_12: int: No parameter help available
- Band_Supported_13: enums.YesNoStatus: No parameter help available
- Power_Class_13: int: No parameter help available
- Band_Supported_14: enums.YesNoStatus: No parameter help available
- Power_Class_14: int: UE power class for band XIV
- Band_Supported_19: enums.YesNoStatus: No parameter help available
- Power_Class_19: int: UE power class for band XIX
- Band_Supported_20: enums.YesNoStatus: No parameter help available
- Power_Class_20: int: No parameter help available
- Band_Supported_21: enums.YesNoStatus: No parameter help available
- Power_Class_21: int: UE power class for band XXI

- Band_Supported_15: enums.YesNoStatus: No parameter help available
- Power_Class_15: int: UE power class for band XV
- Band_Supported_16: enums.YesNoStatus: No parameter help available
- Power_Class_16: int: No parameter help available
- Band_Supported_17: enums.YesNoStatus: No parameter help available
- Power_Class_17: enums.YesNoStatus: No parameter help available
- Band_Supported_18: enums.YesNoStatus: No parameter help available
- Power_Class_18: int: UE power class for band XVIII
- Band_Supported_22: enums.YesNoStatus: No parameter help available
- Power_Class_22: int: UE power class for band XXII
- Band_Supported_25: enums.YesNoStatus: No parameter help available
- Power_Class_25: int: UE power class for band XXV
- Band_Supported_26: enums.YesNoStatus: No parameter help available
- Power_Class_26: int: UE power class for band XXVI
- Band_Supported_32: enums.YesNoStatus: No parameter help available
- Power_Class_32: int: UE power class for band XXXII

get_bc_list() → BcListStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:RFParameter:BCList
value: BcListStruct = driver.sense.ueCapability.rfParameter.get_bc_list()
```

Indicates which band combination the UE supports.

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

get_value() → ValueStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:RFParameter
value: ValueStruct = driver.sense.ueCapability.rfParameter.get_value()
```

Returns RF UE capability information. The value pairs are returned 25 times (band I to XXII, band XXV, XXVI and XXXII) .

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.sense.ueCapability.rfParameter.clone()
```

Subgroups

7.3.3.4.1 Band<Band>

RepCap Settings

```
# Range: B1 .. B32
rc = driver.sense.ueCapability.rfParameter.band.repcap_band_get()
driver.sense.ueCapability.rfParameter.band.repcap_band_set(repcap.Band.B1)
```

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:RFParameter:BAND<Band>
```

class Band

Band commands group definition. 2 total commands, 1 Sub-groups, 1 group commands Repeated Capability: Band, default value after init: Band.B1

class GetStruct

Response structure. Fields:

- Supported: enums.YesNoStatus: NO | YES Support of non-contiguous multi-cell operation
- Power_Class: int: The UE power class
- Add_Sec_Cells: int: Number of additional secondary serving cells supported by the UE. The absence of this IE means that the UE does not support multi-cell operation on three or four cells.
- Ul_Oltd: enums.YesNoStatus: NO | YES Support of uplink open loop transmit diversity
- Nc_2_C: enums.YesNoStatus: NO | YES Support of non-contiguous multi-cell operation on two cells
- Nc_3_C: enums.YesNoStatus: NO | YES Support of non-contiguous multi-cell operation on three cells
- Nc_4_C: enums.YesNoStatus: NO | YES Support of non-contiguous multi-cell operation on four cells
- Ul_Cltd: enums.YesNoStatus: NO | YES Support of uplink closed loop transmit diversity in CELL_DCH
- Ul_Mimo: enums.YesNoStatus: NO | YES Support of uplink MIMO in CELL_DCH
- Mimo_4_X_4_Mode: enums.YesNoStatus: NO | YES Support of MIMO mode with four transmit antennas in CELL_DCH
- Freq_Spec_Cmn_Cop: enums.YesNoStatus: NO | YES Support of frequency-specific compressed mode for intra-band non-contiguous operation

get(*band=<Band.Default: -1>*) → GetStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:RFParameter:BAND<band>
value: GetStruct = driver.sense.ueCapability.rfParameter.band.get(band = repcap.
↪Band.Default)
```

Queries the UE capabilities for the selected band related to non-contiguous multi-cell operation.

param band optional repeated capability selector. Default value: B1 (settable in the interface 'Band')

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.ueCapability.rfParameter.band.clone()
```

Subgroups

7.3.3.4.1.1 Nc<NonContigCell>

RepCap Settings

```
# Range: Nc2 .. Nc4
rc = driver.sense.ueCapability.rfParameter.band.nc.repcap_nonContigCell_get()
driver.sense.ueCapability.rfParameter.band.nc.repcap_nonContigCell_set(repcap.
↪NonContigCell.Nc2)
```

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:RFParameter:BAND<Band>:NC<NonContigCell>
```

class Nc

Nc commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: NonContigCell, default value after init: NonContigCell.Nc2

class GetStruct

Response structure. Fields:

- Supported: enums.YesNoStatus: NO | YES Indicates if the UE supports non-contiguous multi-cell operation for the selected band/cell combination
- Gap_Size: enums.GapSize: M5 | M10 | ANY The maximum gap size between the aggregated cells supported by the UE M5: 5 MHz M10: 10 MHz ANY: any multiple of 5 MHz
- Nc_Comb_22: enums.YesNoStatus: NO | YES Indicates if the UE supports an equal number of contiguous cells on each side of the gap
- Nc_Comb_1331: enums.YesNoStatus: NO | YES Indicates if the UE supports a different number of contiguous cells on each side of the gap

get(*band=<Band.Default: -1>*, *nonContigCell=<NonContigCell.Default: -1>*) → GetStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:RFParameter:BAND<band>:NC
↪<cell>
value: GetStruct = driver.sense.ueCapability.rfParameter.band.nc.get(band =␣
↪repcap.Band.Default, nonContigCell = repcap.NonContigCell.Default)
```

Queries the UE capabilities related to non-contiguous multi-cell operation.

param band optional repeated capability selector. Default value: B1 (settable in the interface 'Band')

param nonContigCell optional repeated capability selector. Default value: Nc2 (settable in the interface 'Nc')

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.ueCapability.rfParameter.band.nc.clone()
```

7.3.3.4.2 Bc<BandCombination>

RepCap Settings

```
# Range: Nr1 .. Nr32
rc = driver.sense.ueCapability.rfParameter.bc.repcap_bandCombination_get()
driver.sense.ueCapability.rfParameter.bc.repcap_bandCombination_set(repcap.
↪BandCombination.Nr1)
```

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UECapability:RFParameter:BC<BandCombination>
```

class Bc

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

class GetStruct

Response structure. Fields:

- Ccomp_12: enums.YesNoStatus: NO | YES Indicates if the UE supports one contiguous carrier in band A and the maximum number of two contiguous carriers in band B
- Ccomp_21: enums.YesNoStatus: NO | YES Indicates if the UE supports the maximum number of two contiguous carriers in band A and one contiguous carrier in band B
- Ccomp_13: enums.YesNoStatus: NO | YES Indicates if the UE supports one contiguous carrier in band A and the maximum number of three contiguous carriers in band B
- Ccomp_31: enums.YesNoStatus: NO | YES Indicates if the UE supports the maximum number of three contiguous carriers in band A and one contiguous carrier in band B
- Ccomp_22: enums.YesNoStatus: NO | YES Indicates if the UE supports the maximum number of two contiguous carriers in band A and the maximum number of two contiguous carriers in band B

get(*bandCombination=<BandCombination.Default: -1>*) → GetStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UECapability:RFParameter:BC<nr>
value: GetStruct = driver.sense.ueCapability.rfParameter.bc.get(bandCombination␣
↪= repcap.BandCombination.Default)
```

Indicates which carrier combination for specific band combination the UE supports.

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

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.ueCapability.rfParameter.bc.clone()
```

7.3.4 UesInfo

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:APN
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:DULalignment
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:DINFo
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:IMEI
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:RIDentity
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:RITYpe
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:TTY
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:CNUMber
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:DNUMber
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:EMERgency
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:ESCategory
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:RRC
```

class UesInfo

UesInfo commands group definition. 15 total commands, 2 Sub-groups, 12 group commands

class DinfoStruct

Structure for reading output parameters. Fields:

- Cmw_Demod_Info: str: 'Uplink Power Underflow': the UL signal power is too low 'Uplink Power in Range': the UL signal power is in range 'Uplink Power Overflow': the UL signal power is too high
- Power_C_1: enums.CellPower: UFL | OK | OFL Cell 1 information: UFL: the UL signal power is too low OK: the UL signal power is in range OFL: the UL signal power is too high
- Sync_C_1: enums.Sync: NOSYnc | OK Cell 1 information: NOSYnc: synchronization to the uplink signal failed OK: successful synchronization to the uplink signal
- Power_C_2: enums.CellPower: UFL | OK | OFL Cell 2 information: UFL: the UL signal power is too low OK: the UL signal power is in range OFL: the UL signal power is too high
- Sync_C_2: enums.Sync: NOSYnc | OK Cell 2 information: NOSYnc: synchronization to the uplink signal failed OK: successful synchronization to the uplink signal

class DulAlignmentStruct

Structure for reading output parameters. Fields:

- Carrier_1: float: Range: 0 chips to 10000 chips, Unit: chips
- Carrier_2: float: Range: 0 chips to 10000 chips, Unit: chips

class EsCategoryStruct

Structure for reading output parameters. Fields:

- Police: bool: OFF | ON OFF: no emergency call to police ON: emergency call to police
- Ambulance: bool: OFF | ON
- Fire_Brigade: bool: OFF | ON
- Marine_Guard: bool: OFF | ON
- Mountain_Rescue: bool: OFF | ON
- Manual: bool: OFF | ON OFF: no emergency calls set up manually ON: emergency calls set up manually
- Automatical: bool: OFF | ON OFF: no emergency calls set up automatically ON: emergency calls set up automatically

get_apn() → List[str]

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:APN
value: List[str] = driver.sense.uesInfo.get_apn()
```

Returns all access point names used by the UE during a packet data connection.

return apn: The names of all connected APNs as a string

get_cnumber() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:CNUMber
value: str = driver.sense.uesInfo.get_cnumber()
```

Queries the calling number for a UE originated call.

return number: Calling number as string with up to 129 digits.

get_dinfo() → DinfoStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:DINFo
value: DinfoStruct = driver.sense.uesInfo.get_dinfo()
```

Queries the demodulation info provided by the demodulator stage of the instrument while it perceives an uplink signal. Information about cell two are relevant only if the dual carrier HSPA scenario is active.

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

get_dnumber() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:DNUMber
value: str = driver.sense.uesInfo.get_dnumber()
```

Queries the number dialed at the UE.

return number: Dialed number as string with up to 129 digits.

get_dul_alignment() → DulAlignmentStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:DULalignment
value: DulAlignmentStruct = driver.sense.uesInfo.get_dul_alignment()
```

Returns the offset between DL DPCH and UL DPCH at the RF connectors of the instrument per carrier.

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

get_emergency() → bool

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:EMERgency
value: bool = driver.sense.uesInfo.get_emergency()
```

Queries whether the established connection is an emergency call.

return active: OFF | ON ON: emergency call OFF: no emergency call

get_es_category() → EsCategoryStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:ESCategory
value: EsCategoryStruct = driver.sense.uesInfo.get_es_category()
```

Returns the service category used during emergency call.

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

get_imei() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:IMEI
value: str = driver.sense.uesInfo.get_imei()
```

Queries the IMEI of the UE.

return imei: IMEI as string with up to 18 digits.

get_ri_type() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:RITYpe
value: str = driver.sense.uesInfo.get_ri_type()
```

Queries the type of the registration identity received from the UE during registration.

return ri_type: 'IMSI' | 'IMEI' | 'IMSISV' | 'TMSI' | 'UNKN' Registration identity type as string. 'UNKN' means unknown.

get_ridentity() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:RIDentity
value: str = driver.sense.uesInfo.get_ridentity()
```

Queries the registration identity received from the UE during registration.

return identity: Registration identity as string with up to 18 digits.

get_rrc() → RsCmwWcdmaSig.enums.RrcState

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:RRC
value: enums.RrcState = driver.sense.uesInfo.get_rrc()
```

Returns the RRC protocol state of the UE.

return state: IDLE | FACH | CPCH | UPCH | DCH Idle mode, CELL_FACH, CELL_PCH, URA_PCH, CELL_DCH

get_tty() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:TTY
value: str = driver.sense.uesInfo.get_tty()
```

Queries whether the UE supports cellular text telephony (CTM) .

return tty: 'supported' | 'not supported' 'supported': CTM supported 'not supported': CTM not supported

Cloning the Group

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

Subgroups

7.3.4.1 UeAddress

class UeAddress

UeAddress 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.uesInfo.ueAddress.clone()
```

Subgroups

7.3.4.1.1 Ipv<IPversion>

RepCap Settings

```
# Range: IPv4 .. IPv6
rc = driver.sense.uesInfo.ueAddress.ipv.repcap_iPversion_get()
driver.sense.uesInfo.ueAddress.ipv.repcap_iPversion_set(repcap.IPversion.IPv4)
```

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:UEADdress:IPV<IPversion>
```

class Ipv

Ipv commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: IPversion, default value after init: IPversion.IPv4

get(*iPversion=<IPversion.Default: -1>*) → List[str]

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:UEADdress:IPV<n>
value: List[str] = driver.sense.uesInfo.ueAddress.ipv.get(iPversion = repcap.
↪IPversion.Default)
```

Returns IPv4 address (<n> = 4) or the IPv6 prefix (<n> = 6) for each APN assigned to the UE by the R&S CMW.

param iPversion optional repeated capability selector. Default value: IPv4 (settable in the interface 'Ipv')

return ip_address: All used IP addresses/prefixes as a string

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uesInfo.ueAddress.ipv.clone()
```

7.3.4.2 Connection

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:CONNection:PACKet
SENSe:WCDMa:SIGNaling<Instance>:UESinfo:CONNection:CIRCuit
```

class Connection

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

get_circuit() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:CONNection:CIRCuit
value: str = driver.sense.uesInfo.connection.get_circuit()
```

Queries the type of an established CS connection. NAV indicates that no CS connection has been established.

return circuit_connect: Connection type as string

get_packet() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UESinfo:CONNection:PACKet
value: str = driver.sense.uesInfo.connection.get_packet()
```

Queries the type of an established PS connection. NAV indicates that no PS connection has been established.

return packet_connect: Connection type as string

7.3.5 Cell

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:CELL:CONFig
```

class Cell

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

get_config() → RsCmwWcdmaSig.enums.CellConfig

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:CELL:CONFig
value: enums.CellConfig = driver.sense.cell.get_config()
```

Returns information corresponding to the gray/green icons displayed behind the cell state in the 'Connection Status' area of the main view. The icons indicate the type of a PS connection.

return config: WCDMa | HSDPa | HSPLus | DCHS | HSPA | HDUPlus | DDUPlus | DHDU | 3CHS | 3DUPlus | 3HDU WCDMa: R99 signal, no HSPA test mode HSDPa: HSDPA HSPLus: HSDPA+ DCHS: dual carrier HSDPA+ HSPA: HSDPA and HSUPA HDUPlus: HSDPA+ and HSUPA DDUPlus: dual carrier HSDPA+ and single carrier HSUPA DHDU: dual carrier HSDPA+ and dual carrier HSUPA 3CHS: three carrier HSDPA+ 3DUPlus: three carrier HSDPA+ and single carrier HSUPA 3HDU: three carrier HSDPA+ and dual carrier HSUPA

7.3.6 IqOut

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:IQOut:CARRier<Carrier>
```

class IqOut

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

class CarrierStruct

Structure for reading output parameters. 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_carrier() → CarrierStruct

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:IQOut:CARRier<carrier>
value: CarrierStruct = driver.sense.iqOut.get_carrier()
```

Queries properties of the baseband signal at the I/Q output.

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

Global Repeated Capabilities: repcap.Carrier

7.3.7 Downlink

class Downlink

Downlink 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.downlink.clone()
```

Subgroups

7.3.7.1 Carrier

class Carrier

Carrier 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.downlink.carrier.clone()
```

Subgroups

7.3.7.1.1 Enhanced

class Enhanced

Enhanced 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.downlink.carrier.enhanced.clone()
```

Subgroups

7.3.7.1.1.1 Dpch

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:DL:CARRier<Carrier>:ENHanced:DPCH:REPorted
```

class Dpch

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

get_reported() → float

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:DL:CARRier<carrier>
↪:ENHanced:DPCH:REPorted
value: float = driver.sense.downlink.carrier.enhanced.dpch.get_reported()
```

Displays the downlink DPCH/F-DPCH level reported by the UE.

return level: DPCH/F-DPCH level relative to the base level Ior Range: -80 dB to 0 dB, Unit: dB

Global Repeated Capabilities: repcap.Carrier

7.3.8 Uplink

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UL:EIPower
```

class Uplink

Uplink commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

get_ei_power() → float

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UL:EIPower
value: float = driver.sense.uplink.get_ei_power()
```

Queries the expected initial DPCCH power.

return exp_dpcchp_ower: Range: -160 dBm to 33 dBm, Unit: dBm

Cloning the Group

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

Subgroups

7.3.8.1 OlpControl

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:UL:OLPControl:EIPPower
```

class OlpControl

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

get_eip_power() → float

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:UL:OLPControl:EIPPower
value: float = driver.sense.uplink.olpControl.get_eip_power()
```

Queries the expected initial preamble power.

return exp_preamble_pwr: Range: -160 dBm to 33 dBm, Unit: dBm

7.3.9 Connection

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:CONNection:CURRent
```

class Connection

Connection commands group definition. 3 total commands, 1 Sub-groups, 1 group commands

get_current() → RsCmwWcdmaSig.enums.CurrentConnectionType

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:CONNection:CURRent
value: enums.CurrentConnectionType = driver.sense.connection.get_current()
```

Queries the type of the current connection.

return type_py: NONE | VOICe | VIDeo | SRB | TEST | PACKet NONE: none active connection VOICe: voice connection VIDeo: video connection SRB: signaling radio bearer only TEST: test mode PACKet: packet data connection using DAU

Cloning the Group

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

Subgroups

7.3.9.1 Cswitched

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:CONNection:CSWitched:ATTempt
SENSe:WCDMa:SIGNaling<Instance>:CONNection:CSWitched:REJect
```

class Cswitched

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

get_attempt() → int

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:CONNection:CSWitched:ATTempt
value: int = driver.sense.connection.cswitched.get_attempt()
```

Queries the counters of connection attempt / reject.

return counter: Range: 0 to 2^32

get_reject() → int

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:CONNection:CSWitched:REJect
value: int = driver.sense.connection.cswitched.get_reject()
```

Queries the counters of connection attempt / reject.

return counter: Range: 0 to 2^32

7.3.10 Sms

class Sms

Sms 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.sense.sms.clone()
```

Subgroups

7.3.10.1 Outgoing

class Outgoing

Outgoing 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.sms.outgoing.clone()
```

Subgroups

7.3.10.1.1 Info

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:SMS:OUTGoing:INFO:LMSent
```

class Info

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

get_lmsent() → RsCmwWcdmaSig.enums.SucessState

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:SMS:OUTGoing:INFO:LMSent
value: enums.SucessState = driver.sense.sms.outgoing.info.get_lmsent()
```

Indicates, whether the last message was sent successfully or not.

return state: FAILed | SUCCessful

7.3.10.2 Incoming

class Incoming

Incoming 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.sense.sms.incoming.clone()
```

Subgroups

7.3.10.2.1 Info

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:SMS:INComing:INFO:MTEXt
SENSe:WCDMa:SIGNaling<Instance>:SMS:INComing:INFO:MLENgth
```

class Info

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

get_mlength() → int

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:SMS:INComing:INFO:MLENgth
value: int = driver.sense.sms.incoming.info.get_mlength()
```

Returns the length of the last SMS message received from the UE.

return message_length: Number of characters of the message Range: 0 to 160

get_mtext() → str

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:SMS:INComing:INFO:MTEXt
value: str = driver.sense.sms.incoming.info.get_mtext()
```

Returns the text of the last SMS message received from the UE. Only 7-bit ASCII text is supported.

return message_text: Message text as string

7.3.10.3 Info

class Info

Info 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.sms.info.clone()
```

Subgroups

7.3.10.3.1 LrMessage

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:SMS:INFO:LRMessage:RFLag
```

class LrMessage

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

get_rflag() → bool

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:SMS:INFO:LRMessage:RFLag
value: bool = driver.sense.sms.info.lrMessage.get_rflag()
```

Queries the 'message read' flag for the last received message. INTRO_CMD_HELP: The flag is true (ON) in the following cases:

- No SMS message has been received.
- The last received SMS message has been read, see method RsCmwWcdmaSig.Sense.Sms.Incoming.Info.mtext.

- The last received SMS message has been deleted, see method RsCmwWcdmaSig.Clean.Sms.Incoming.Info.Mtext.set.

return last_rec_mess_read: OFF | ON OFF: unread message available ON: no unread message available

7.3.11 Fading

class Fading
Fading 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.fading.clone()
```

Subgroups

7.3.11.1 Carrier

class Carrier
Carrier 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.fading.carrier.clone()
```

Subgroups

7.3.11.1.1 Fsimulator

class Fsimulator
Fsimulator 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.fading.carrier.fsimulator.clone()
```

Subgroups

7.3.11.1.1.1 Iloss

SCPI Commands

```
SENSe:WCDMa:SIGNaling<Instance>:FADing:CARRier<Carrier>:FSIMulator:ILOSs:CSAMples
```

class Iloss

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

get_csamples() → float

```
# SCPI: SENSe:WCDMa:SIGNaling<instance>:FADing:CARRier<carrier>
↪:FSIMulator:ILOSs:CSAMples
value: float = driver.sense.fading.carrier.fsimulator.iloss.get_csamples()
```

Displays the percentage of clipped samples.

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

Global Repeated Capabilities: repcap.Carrier

7.4 Clean

class Clean

Clean 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.clean.clone()
```

Subgroups

7.4.1 Elogging

SCPI Commands

```
CLEan:WCDMa:SIGNaling<Instance>:ELOGging
```

class Elogging

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

set() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:ELOGging
driver.clean.elogging.set()
```

Clears the event log.

set_with_opc() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:ELOGging
driver.clean.elogging.set_with_opc()
```

Clears the event log.

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

7.4.2 Connection

class Connection

Connection 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.clean.connection.clone()
```

Subgroups

7.4.2.1 Cswitched

class Cswitched

Cswitched 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.clean.connection.cswitched.clone()
```

Subgroups

7.4.2.1.1 Attempt

SCPI Commands

```
CLEan:WCDMa:SIGNaling<Instance>:CONNection:CSWitched:ATTempt
```

class Attempt

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

set() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:CONNection:CSWitched:ATTempt
driver.clean.connection.cswitched.attempt.set()
```

Sets the counters of connection attempt / reject to zero.

set_with_opc() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:CONNection:CSWitched:ATTempt
driver.clean.connection.cswitched.attempt.set_with_opc()
```

Sets the counters of connection attempt / reject to zero.

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

7.4.2.1.2 Reject

SCPI Commands

```
CLEan:WCDMa:SIGNaling<Instance>:CONNection:CSWitched:REJect
```

class Reject

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

set() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:CONNection:CSWitched:REJect
driver.clean.connection.cswitched.reject.set()
```

Sets the counters of connection attempt / reject to zero.

set_with_opc() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:CONNection:CSWitched:REJect
driver.clean.connection.cswitched.reject.set_with_opc()
```

Sets the counters of connection attempt / reject to zero.

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

7.4.3 Sms

class Sms

Sms 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.clean.sms.clone()
```

Subgroups

7.4.3.1 Incoming

class Incoming
Incoming 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.clean.sms.incoming.clone()
```

Subgroups

7.4.3.1.1 Info

class Info
Info 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.clean.sms.incoming.info.clone()
```

Subgroups

7.4.3.1.1.1 Mtext

SCPI Commands

```
CLEan:WCDMa:SIGNaling<Instance>:SMS:INComing:INFO:MTEXt
```

class Mtext
Mtext commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

set() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:SMS:INComing:INFO:MTEXt
driver.clean.sms.incoming.info.mtext.set()
```

Resets all parameters related to a received SMS message. The message text and the information about the message length are deleted. The 'message read' flag is set to true.

set_with_opc() → None

```
# SCPI: CLEan:WCDMa:SIGNaling<instance>:SMS:INComing:INFO:MTEXt
driver.clean.sms.incoming.info.mtext.set_with_opc()
```

Resets all parameters related to a received SMS message. The message text and the information about the message length are deleted. The 'message read' flag is set to true.

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

7.5 UeReport

class UeReport

UeReport 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.ueReport.clone()
```

Subgroups

7.5.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:UEReport:STATe
```

class State

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

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:UEReport:STATe
value: enums.ResourceState = driver.ueReport.state.fetch()
```

Queries the state of UE measurement reporting.

return state: RDY | PENDing RDY: Any requested reports have been received. PENDing: The instrument is waiting for reports from the UE.

7.6 Route

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>
```

class Route

Route commands group definition. 34 total commands, 1 Sub-groups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SCELl | DCARrier | SCFading | DCFading | SCFDiversity | DCFDiversity | DBFading | DBFDiversity | DCHSpa | TCHSpa SCEL: 'Standard Cell' DCARrier: 'Dual Carrier' SCFading: 'Standard Cell Fading' DCFading: 'Dual Carrier Fading' SCFDiversity: 'Standard Cell RX Diversity Fading' DCFDiversity: 'Dual Carrier RX Diversity Fading' DBFading: 'Dual Carrier / Dual Band Fading' DBFDiversity: 'Dual Carrier / Dual Band RX Diversity Fading' DCHSpa: 'Dual Carrier HSPA' TCHSpa: '3C HSPA'
- Master: str: For future use - returned value not relevant
- Rx_Connector: enums.RxConnector: RF connector for input path 1
- Rx_Converter: enums.RxConverter: RX module for input path 1
- Rx_2_Connector: enums.RxConnector: RF connector for input path 2
- Rx_2_Converter: enums.RxConverter: RX module for input path 2
- Tx_Connector: enums.TxConnector: RF connector for output path 1
- Tx_Converter: enums.TxConverter: TX module for output path 1
- Tx_2_Connector: enums.TxConnector: RF connector for output path 2
- Tx_2_Converter: enums.TxConverter: TX module for output path 2
- Tx_3_Connector: enums.TxConnector: RF connector for output path 2
- Tx_3_Converter: enums.TxConverter: TX module for output path 2
- Tx_4_Connector: enums.TxConnector: RF connector for output path 3
- Tx_4_Converter: enums.TxConverter: TX module for output path 4
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for output path 1
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for output path 2
- Iq_3_Connector: enums.TxConnector: DIG IQ OUT connector for output path 3
- Iq_4_Connector: enums.TxConnector: DIG IQ OUT connector for output path 4
- Fader: enums.Fader: I/Q board with I/Q connectors

get_value() → ValueStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>
value: ValueStruct = driver.route.get_value()
```

Returns the configured routing settings. The number of returned values depends on the active scenario (6 to 18 values) . 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

7.6.1 Scenario

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario
```

class Scenario

Scenario commands group definition. 33 total commands, 10 Sub-groups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SCELl | DCARrier | SCFading | DCFading | SCFDiversity | DCFDiversity | DBFading | DBFDiversity | DCHSpa | TCHSpa SCEL: 'Standard Cell' DCARrier: 'Dual Carrier' SCFading: 'Standard Cell Fading' DCFading: 'Dual Carrier Fading' SCFDiversity:'Standard Cell RX Diversity Fading' DCFDiversity: 'Dual Carrier RX Diversity Fading' DBFading: 'Dual Carrier / Dual Band Fading' DBFDiversity: 'Dual Carrier / Dual Band RX Diversity Fading' DCHSpa: 'Dual Carrier HSPA' TCHSpa: '3C HSPA'
- Fader: enums.SourceInt: EXTernal | INTernal Only returned for internal fading scenarios, e.g. SCF, DCF Indicates whether internal or external fading is active.

get_value() → ValueStruct

```
# SCPI: ROUTe:WCDMa: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.

Cloning the Group

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

Subgroups

7.6.1.1 Scell

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCELl:FLEXible
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCELl
```

class Scell

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

class FlexibleStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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:

- 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_flexible() → FlexibleStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCELl:FLEXible
value: FlexibleStruct = driver.route.scenario.scell.get_flexible()
```

Activates the standard cell 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 FlexibleStruct structure arguments.

get_value() → ValueStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCELl
value: ValueStruct = driver.route.scenario.scell.get_value()
```

Activates the standard cell 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 ValueStruct structure arguments.

set_flexible(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Scell.Scell.FlexibleStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCELl:FLEXible
driver.route.scenario.scell.set_flexible(value = FlexibleStruct())
```

Activates the standard cell scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for FlexibleStruct structure arguments.

set_value(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Scell.Scell.ValueStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCELl
driver.route.scenario.scell.set_value(value = ValueStruct())
```

Activates the standard cell scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ValueStruct structure arguments.

7.6.1.2 Dcarrier

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCARrier:FLEXible
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCARrier
```

class Dcarrier

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

class FlexibleStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Multi-band operation requires different modules for the two paths.

class ValueStruct

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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Multi-band operation requires different modules for the two paths.

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCARrier:FLEXible
value: FlexibleStruct = driver.route.scenario.dcarrier.get_flexible()
```

Activates the scenario 'Dual Carrier' 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 FlexibleStruct structure arguments.

get_value() → ValueStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCARrier
value: ValueStruct = driver.route.scenario.dcarrier.get_value()
```

Activates the scenario 'Dual Carrier' 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 ValueStruct structure arguments.

set_flexible(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Dcarrier.Dcarrier.FlexibleStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCARrier:FLEXible
driver.route.scenario.dcarrier.set_flexible(value = FlexibleStruct())
```

Activates the scenario 'Dual Carrier' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for FlexibleStruct structure arguments.

set_value(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Dcarrier.Dcarrier.ValueStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCARrier
driver.route.scenario.dcarrier.set_value(value = ValueStruct())
```

Activates the scenario 'Dual Carrier' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ValueStruct structure arguments.

7.6.1.3 ScFading

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFading:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFading:INTernal
```

class ScFading

ScFading commands group definition. 4 total commands, 1 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
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa: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:WCDMa: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. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScFading.ScFading.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa: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:*
RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScFading.ScFading.InternalStruct) → None

```
# SCPI: ROUTe:WCDMa: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. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

Cloning the Group

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

Subgroups

7.6.1.3.1 Flexible

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFading:FLEXible:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFading:FLEXible:INTernal
```

class Flexible

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCFading:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.scFading.flexible.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:WCDMa:SIGNaling<instance>:SCENario:SCFading:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.scFading.flexible.get_internal()
```

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScFading_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCFading:FLEXible[:EXTernal]
driver.route.scenario.scFading.flexible.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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScFading_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCFading:FLEXible:INTernal
driver.route.scenario.scFading.flexible.set_internal(value = InternalStruct())
```

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.6.1.4 ScfDiversity

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFDiversity:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFDiversity:INTernal
```

class ScfDiversity

ScfDiversity commands group definition. 4 total commands, 1 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path.
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path. Select different connectors for the two paths.

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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCFDiversity[:EXTernal]
value: ExternalStruct = driver.route.scenario.scfDiversity.get_external()
```

Activates the 'Standard Cell RX Diversity 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:WCDMa:SIGNaling<instance>:SCENario:SCFDiversity:INTernal
value: InternalStruct = driver.route.scenario.scfDiversity.get_internal()
```

Activates the 'Standard Cell RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScfDiversity.ScfDiversity.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCFDiversity[:EXTernal]
driver.route.scenario.scfDiversity.set_external(value = ExternalStruct())
```

Activates the 'Standard Cell RX Diversity 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScfDiversity.ScfDiversity.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCFDiversity:INTernal
driver.route.scenario.scfDiversity.set_internal(value = InternalStruct())
```

Activates the 'Standard Cell RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

Cloning the Group

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

Subgroups

7.6.1.4.1 Flexible

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible:INTernal
```

class Flexible

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path.
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path. Select different connectors for the two paths.

class InternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling path
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>
↪:SCENario:SCFDiversity:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.scfDiversity.flexible.get_
↪external()
```

Activates the 'Standard Cell RX Diversity 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:WCDMa:SIGNaling<instance>:SCENario:SCFDiversity:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.scfDiversity.flexible.get_
↪internal()
```

Activates the 'Standard Cell RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScfDiversity_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>
↪:SCENario:SCFDiversity:FLEXible[:EXTernal]
driver.route.scenario.scfDiversity.flexible.set_external(value =␣
↪ExternalStruct())
```

Activates the 'Standard Cell RX Diversity 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.ScfDiversity_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:SCFDiversity:FLEXible:INTernal
driver.route.scenario.scfDiversity.flexible.set_internal(value =␣
↪InternalStruct())
```

Activates the 'Standard Cell RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.6.1.5 DcFading

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFading:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFading:INTernal
```

class DcFading

DcFading commands group definition. 4 total commands, 1 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path. Select different connectors for the two paths.

- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFading[:EXTernal]
value: ExternalStruct = driver.route.scenario.dcFading.get_external()
```

Activates the 'Dual Carrier 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:WCDMa:SIGNaling<instance>:SCENario:DCFading:INTernal
value: InternalStruct = driver.route.scenario.dcFading.get_internal()
```

Activates the 'Dual Carrier Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcFading.DcFading.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFading[:EXTernal]
driver.route.scenario.dcFading.set_external(value = ExternalStruct())
```

Activates the 'Dual Carrier 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcFading.DcFading.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFading:INTernal
driver.route.scenario.dcFading.set_internal(value = InternalStruct())
```

Activates the 'Dual Carrier Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

Cloning the Group

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

Subgroups

7.6.1.5.1 Flexible

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFading:FLEXible:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFading:FLEXible:INTernal
```

class Flexible

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path. Select different connectors for the two paths.
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path

class InternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFading:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.dcFading.flexible.get_external()
```

Activates the 'Dual Carrier 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:WCDMa:SIGNaling<instance>:SCENario:DCFading:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.dcFading.flexible.get_internal()
```

Activates the 'Dual Carrier Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcFading_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFading:FLEXible[:EXTernal]
driver.route.scenario.dcFading.flexible.set_external(value = ExternalStruct())
```

Activates the 'Dual Carrier 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcFading_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFading:FLEXible:INTernal
driver.route.scenario.dcFading.flexible.set_internal(value = InternalStruct())
```

Activates the 'Dual Carrier Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.6.1.6 DcfDiversity

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFDiversity:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFDiversity:INTernal
```

class DcfDiversity

DcfDiversity commands group definition. 4 total commands, 1 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path. Select different connectors for the two paths.
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Select different modules for the two paths.
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFDiversity[:EXTernal]
value: ExternalStruct = driver.route.scenario.dcfDiversity.get_external()
```

Activates the 'Dual Carrier RX Diversity 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:WCDMa:SIGNaling<instance>:SCENario:DCFDiversity:INTernal
value: InternalStruct = driver.route.scenario.dcfDiversity.get_internal()
```

Activates the 'Dual Carrier RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcfDiversity.DcfDiversity.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFDiversity[:EXTernal]
driver.route.scenario.dcfDiversity.set_external(value = ExternalStruct())
```

Activates the 'Dual Carrier RX Diversity 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcfDiversity.DcfDiversity.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFDiversity:INTernal
driver.route.scenario.dcfDiversity.set_internal(value = InternalStruct())
```

Activates the 'Dual Carrier RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

Cloning the Group

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

Subgroups

7.6.1.6.1 Flexible

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFDiversity:FLEXible:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCFDiversity:FLEXible:INTernal
```

class Flexible

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path. Select different connectors for the two paths.
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path

class InternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Select different modules for the two paths.
- Fader: enums.Fader: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>
↪:SCENario:DCFDiversity:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.dcfDiversity.flexible.get_
↪external()
```

Activates the 'Dual Carrier RX Diversity 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:WCDMa:SIGNaling<instance>:SCENario:DCFDiversity:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.dcfDiversity.flexible.get_
↪internal()
```

Activates the 'Dual Carrier RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcfDiversity_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>
↪:SCENario:DCFDiversity:FLEXible[:EXTernal]
driver.route.scenario.dcfDiversity.flexible.set_external(value =␣
↪ExternalStruct())
```

Activates the 'Dual Carrier RX Diversity 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DcfDiversity_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCFDiversity:FLEXible:INTernal
driver.route.scenario.dcfDiversity.flexible.set_internal(value =␣
↪InternalStruct())
```

Activates the 'Dual Carrier RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.6.1.7 DbFading

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFading:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFading:INTernal
```

class DbFading

DbFading commands group definition. 4 total commands, 1 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path. Select different connectors for the two paths.

- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Fader_1: enums.Fader: Internal fader used for the output path of carrier one.
- Fader_2: enums.Fader: Internal fader used for the output path of carrier two. Select different faders for the two carriers.

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFading[:EXTernal]
value: ExternalStruct = driver.route.scenario.dbFading.get_external()
```

Activates the 'Dual Carrier / Dual Band 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:WCDMa:SIGNaling<instance>:SCENario:DBFading:INTernal
value: InternalStruct = driver.route.scenario.dbFading.get_internal()
```

Activates the 'Dual Carrier / Dual Band Fading: Internal' scenario and selects the signal paths. To set the signaling unit manually, use the command method RsCmwWcdmaSig.Route.Scenario.DbFading.Flexible.internal instead. 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbFading.DbFading.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFading[:EXTernal]
driver.route.scenario.dbFading.set_external(value = ExternalStruct())
```

Activates the 'Dual Carrier / Dual Band 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:*
RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbFading.DbFading.InternalStruct)
→ None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFading:INTernal
driver.route.scenario.dbFading.set_internal(value = InternalStruct())
```

Activates the 'Dual Carrier / Dual Band Fading: Internal' scenario and selects the signal paths. To set the signaling unit manually, use the command method RsCmwWcdmaSig.Route.Scenario.DbFading.Flexible.internal instead. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

Cloning the Group

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

Subgroups

7.6.1.7.1 Flexible

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFading:FLEXible:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFading:FLEXible:INTernal
```

class Flexible

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path. Select different connectors for the two paths.
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path

class InternalStruct
Structure for reading output parameters. Fields:

- Bb_Board: enums.BasebandBoard: Signaling unit
- 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 first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Fader_1: enums.Fader: Internal fader used for the output path of carrier one.
- Fader_2: enums.Fader: Internal fader used for the output path of carrier two. Select different faders for the two carriers.

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFading:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.dbFading.flexible.get_external()
```

Activates the 'Dual Carrier / Dual Band 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:WCDMa:SIGNaling<instance>:SCENario:DBFading:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.dbFading.flexible.get_internal()
```

Activates the 'Dual Carrier / Dual Band Fading: Internal' 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 InternalStruct structure arguments.

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbFading_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFading:FLEXible[:EXTernal]
driver.route.scenario.dbFading.flexible.set_external(value = ExternalStruct())
```

Activates the 'Dual Carrier / Dual Band 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbFading_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFading:FLEXible:INTernal
driver.route.scenario.dbFading.flexible.set_internal(value = InternalStruct())
```

Activates the 'Dual Carrier / Dual Band Fading: Internal' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.6.1.8 DbfDiversity

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFDiversity:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFDiversity:INTernal
```

class DbfDiversity

DbfDiversity commands group definition. 4 total commands, 1 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 first output path of the carrier one
- Tx_Converter: enums.TxConverter: TX module for the first output path of the carrier one. Select different modules for each of the paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path of the carrier one
- Tx_2_Converter: enums.TxConverter: TX module for the second output path of the carrier one
- Tx_3_Connector: enums.TxConnector: RF connector for the first output path of the carrier two
- Tx_3_Converter: enums.TxConverter: TX module for the first output path of the carrier two
- Tx_4_Connector: enums.TxConnector: RF connector for the second output path of the carrier two
- Tx_4_Converter: enums.TxConverter: TX module for the second output path of the carrier two
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path of the carrier one
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path of the carrier one
- Iq_3_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path of the carrier two
- Iq_4_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path of the carrier two

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 first output path of the carrier one

- Tx_Converter: enums.TxConverter: TX module for the first output path of the carrier one. Select different modules for each of the paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path of the carrier one
- Tx_2_Converter: enums.TxConverter: TX module for the second output path of the carrier one
- Tx_3_Connector: enums.TxConnector: RF connector for the first output path of the carrier two
- Tx_3_Converter: enums.TxConverter: TX module for the first output path of the carrier two
- Tx_4_Connector: enums.TxConnector: RF connector for the second output path of the carrier two
- Tx_4_Converter: enums.TxConverter: TX module for the second output path of the carrier two
- Fader_1: enums.Fader: Internal fader used for the second output path of carrier one.
- Fader_2: enums.Fader: Internal fader used for the second output path of carrier two. Select different boards for the two carriers.

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFDiversity[:EXTernal]
value: ExternalStruct = driver.route.scenario.dbfDiversity.get_external()
```

Activates the 'Dual Carrier / Dual Band RX Diversity 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:WCDMa:SIGNaling<instance>:SCENario:DBFDiversity:INTernal
value: InternalStruct = driver.route.scenario.dbfDiversity.get_internal()
```

Activates the 'Dual Carrier / Dual Band RX Diversity Fading: Internal' scenario and selects the signal paths. To set the signaling unit manually, use the command method RsCmwWcdmaSig.Route.Scenario.DbfDiversity.Flexible.internal instead. 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbfDiversity.DbfDiversity.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFDiversity[:EXTernal]
driver.route.scenario.dbfDiversity.set_external(value = ExternalStruct())
```

Activates the 'Dual Carrier / Dual Band RX Diversity 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbfDiversity.DbfDiversity.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFDiversity:INTernal
driver.route.scenario.dbfDiversity.set_internal(value = InternalStruct())
```

Activates the 'Dual Carrier / Dual Band RX Diversity Fading: Internal' scenario and selects the signal paths. To set the signaling unit manually, use the command method RsCmwWcdmaSig.Route.Scenario.DbfDiversity.Flexible.internal instead. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

Cloning the Group

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

Subgroups

7.6.1.8.1 Flexible

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFDiversity:FLEXible:EXTernal
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DBFDiversity:FLEXible:INTernal
```

class Flexible

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Bb_Board_1: enums.BasebandBoard: First signaling unit
- Bb_Board_2: enums.BasebandBoard: Second signaling unit
- 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 first output path of the carrier one
- Tx_Converter: enums.TxConverter: TX module for the first output path of the carrier one. Select different modules for each of the paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path of the carrier one
- Tx_2_Converter: enums.TxConverter: TX module for the second output path of the carrier one
- Tx_3_Connector: enums.TxConnector: RF connector for the first output path of the carrier two
- Tx_3_Converter: enums.TxConverter: TX module for the first output path of the carrier two
- Tx_4_Connector: enums.TxConnector: RF connector for the second output path of the carrier two
- Tx_4_Converter: enums.TxConverter: TX module for the second output path of the carrier two
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path of the carrier one

- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path of the carrier one
- Iq_3_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path of the carrier two
- Iq_4_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path of the carrier two

class InternalStruct

Structure for reading output parameters. Fields:

- Bb_Board_1: enums.BasebandBoard: First signaling unit
- Bb_Board_2: enums.BasebandBoard: Second signaling unit
- 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 first output path of the carrier one
- Tx_Converter: enums.TxConverter: TX module for the first output path of the carrier one. Select different modules for each of the paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path of the carrier one
- Tx_2_Converter: enums.TxConverter: TX module for the second output path of the carrier one
- Tx_3_Connector: enums.TxConnector: RF connector for the first output path of the carrier two
- Tx_3_Converter: enums.TxConverter: TX module for the first output path of the carrier two
- Tx_4_Connector: enums.TxConnector: RF connector for the second output path of the carrier two
- Tx_4_Converter: enums.TxConverter: TX module for the second output path of the carrier two
- Fader_1: enums.Fader: Internal fader used for the second output path of carrier one.
- Fader_2: enums.Fader: Internal fader used for the second output path of carrier two. Select different boards for the two carriers.

get_external() → ExternalStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>
↪:SCENario:DBFDiversity:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.dbfDiversity.flexible.get_
↪external()
```

Activates the 'Dual Carrier / Dual Band RX Diversity 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:WCDMa:SIGNaling<instance>:SCENario:DBFDiversity:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.dbfDiversity.flexible.get_
↪internal()
```

Activates the 'Dual Carrier / Dual Band RX Diversity Fading: Internal' 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 InternalStruct structure arguments.

set_external(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbfDiversity_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>
↪:SCENario:DBFDiversity:FLEXible[:EXTernal]
driver.route.scenario.dbfDiversity.flexible.set_external(value =␣
↪ExternalStruct())
```

Activates the 'Dual Carrier / Dual Band RX Diversity 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: RsCmwWcdmaSig.Implementations.Route_.Scenario_.DbfDiversity_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DBFDiversity:FLEXible:INTernal
driver.route.scenario.dbfDiversity.flexible.set_internal(value =␣
↪InternalStruct())
```

Activates the 'Dual Carrier / Dual Band RX Diversity Fading: Internal' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.6.1.9 Dchspa

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCHSpa:FLEXible
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:DCHSpa
```

class Dchspa

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

class FlexibleStruct

Structure for reading output parameters. Fields:

- Bb_Board_1: enums.BasebandBoard: First signaling unit
- Bb_Board_2: enums.BasebandBoard: Second signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the first input path
- Rx_Converter: enums.RxConverter: RX module for the first input path
- Rx_2_Connector: enums.RxConnector: RF connector for the second input path
- Rx_2_Converter: enums.RxConverter: TX module for the second input path. Multi-band operation requires different modules for the two paths.
- Tx_Connector: enums.TxConnector: RF connector for the first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path

- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Multi-band operation requires different modules for the two paths.

class ValueStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the first input path
- Rx_Converter: enums.RxConverter: RX module for the first input path
- Rx_2_Connector: enums.RxConnector: RF connector for the second input path
- Rx_2_Converter: enums.RxConverter: TX module for the second input path. Multi-band operation requires different modules for the two paths.
- Tx_Connector: enums.TxConnector: RF connector for the first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Multi-band operation requires different modules for the two paths.

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCHSpa:FLEXible
value: FlexibleStruct = driver.route.scenario.dchspa.get_flexible()
```

Activates the scenario 'Dual Carrier HSPA' 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 FlexibleStruct structure arguments.

get_value() → ValueStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCHSpa
value: ValueStruct = driver.route.scenario.dchspa.get_value()
```

Activates the scenario 'Dual Carrier HSPA' 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 ValueStruct structure arguments.

set_flexible(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Dchspa.Dchspa.FlexibleStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCHSpa:FLEXible
driver.route.scenario.dchspa.set_flexible(value = FlexibleStruct())
```

Activates the scenario 'Dual Carrier HSPA' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for FlexibleStruct structure arguments.

set_value(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Dchspa.Dchspa.ValueStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:DCHSpa
driver.route.scenario.dchspa.set_value(value = ValueStruct())
```

Activates the scenario 'Dual Carrier HSPA' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ValueStruct structure arguments.

7.6.1.10 Tchspa

SCPI Commands

```
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:TCHSpa:FLEXible
ROUTe:WCDMa:SIGNaling<Instance>:SCENario:TCHSpa
```

class Tchspa

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

class FlexibleStruct

Structure for reading output parameters. Fields:

- Bb_Board_1: enums.BasebandBoard: First signaling unit
- Bb_Board_2: enums.BasebandBoard: Second signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the first input path
- Rx_Converter: enums.RxConverter: RX module for the first input path
- Rx_2_Connector: enums.RxConnector: RF connector for the second input path
- Rx_2_Converter: enums.RxConverter: RX module for the second input path.
- Tx_Connector: enums.TxConnector: RF connector for the first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Select different modules for the path one and two.
- Tx_3_Connector: enums.TxConnector: RF connector for the third output path
- Tx_3_Converter: enums.TxConverter: TX module for the third output path. Select different modules for the three paths or use the path one.

class ValueStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the first input path
- Rx_Converter: enums.RxConverter: RX module for the first input path
- Rx_2_Connector: enums.RxConnector: RF connector for the second input path
- Rx_2_Converter: enums.RxConverter: RX module for the second input path.
- Tx_Connector: enums.TxConnector: RF connector for the first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path

- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Select different modules for the path one and two.
- Tx_3_Connector: enums.TxConnector: RF connector for the third output path
- Tx_3_Converter: enums.TxConverter: TX module for the third output path. Select different modules for the three paths or use the path one.

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:TCHSpa:FLEXible
value: FlexibleStruct = driver.route.scenario.tchspa.get_flexible()
```

Activates the scenario '3C HSPA' 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 FlexibleStruct structure arguments.

get_value() → ValueStruct

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:TCHSpa
value: ValueStruct = driver.route.scenario.tchspa.get_value()
```

Activates the scenario '3C HSPA' 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 ValueStruct structure arguments.

set_flexible(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Tchspa.Tchspa.FlexibleStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:TCHSpa:FLEXible
driver.route.scenario.tchspa.set_flexible(value = FlexibleStruct())
```

Activates the scenario '3C HSPA' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for FlexibleStruct structure arguments.

set_value(*value: RsCmwWcdmaSig.Implementations.Route_.Scenario_.Tchspa.Tchspa.ValueStruct*) → None

```
# SCPI: ROUTe:WCDMa:SIGNaling<instance>:SCENario:TCHSpa
driver.route.scenario.tchspa.set_value(value = ValueStruct())
```

Activates the scenario '3C HSPA' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ValueStruct structure arguments.

7.7 Rsignaling

class Rsignaling
Rsignaling 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.rsignaling.clone()
```

Subgroups

7.7.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:RSIGnaling:STATe
```

class State
State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ReducedSignState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:RSIGnaling:STATe
value: enums.ReducedSignState = driver.rsignaling.state.fetch()
```

Queries the reduced signaling connection state, see also 'Connection States for Reduced Signaling'.

return rsig_state: OFF | PROCessing | ON OFF: reduced signaling Off ON: reduced signaling On PROCessing: switching channels On/Off

7.8 Pswitched

class Pswitched
Pswitched 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.pswitched.clone()
```

Subgroups

7.8.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:PSWitched:STATe
```

class State

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

fetch() → RsCmwWcdmaSig.enums.PswitchedState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:PSWitched:STATe
value: enums.PswitchedState = driver.pswitched.state.fetch()
```

Queries the PS connection state, see also 'PS Connection States'.

return ps_state: OFF | ON | ATTached | CESTablished | RELeasing | CONNecting | SIGNaling | IHPReparate | IHANdover | OHANdover | IRPReparate | IREDirection | OREDirection OFF: signal is off ON: signal is on ATTached: attached CESTablished: connection established RELeasing: disconnect in progress CONNecting: connection setup in progress SIGNaling: signaling in progress IHPReparate: preparation for incoming handover IHANdover incoming handover in progress OHANdover outgoing handover in progress IRPReparate: preparation for incoming redirection IREDirection: incoming redirection in progress OREDirection: outgoing redirection in progress

7.9 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

7.9.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:CSWitched:STATe
```

class State

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

fetch() → RsCmwWcdmaSig.enums.CswitchedState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:CSWitched:STATe
value: enums.CswitchedState = driver.cswitched.state.fetch()
```

Queries the CS connection state, see also 'CS Connection States'. Use method RsCmwWcdmaSig.Call.Cswitched. action to initiate a transition between different connection states. The CS state changes to ON when the signaling generator is started (see method RsCmwWcdmaSig.Source.Cell.State.value) . To make sure that a WCDMA cell signal is available, query the cell state. It must be ON, ADJ (see method RsCmwWcdmaSig.Source.Cell.State.all) .

return cs_state: ON | REGister | ALERting | CONNecting | PAGing | RELeasing | SIGNaling | IHPReparate | IHANdover | OHANdover | OFF | CESTablished | IRPReparate | IREDirection | OREDirection ON: signal is on REGister: registered ALERting: alerting CONNecting: call setup in progress PAGing: paging in progress RELeasing: disconnect in progress SIGNaling: signaling in progress IHPReparate: preparation for incoming handover IHANdover: incoming handover in progress OHANdover: outgoing handover in progress OFF: signal is off CESTablished: call established IRPReparate: preparation for incoming redirection IREDirection: incoming redirection in progress OREDirection: outgoing redirection in progress

7.10 Call

class Call

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

Cloning the Group

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

Subgroups

7.10.1 Rsignaling

SCPI Commands

```
CALL:WCDMa:SIGNaling<Instance>:RSIGnaling:ACTion
```

class Rsignaling

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

set_action(*rs_action: bool*) → None

```
# SCPI: CALL:WCDMa:SIGNaling<instance>:RSIGnaling:ACTion
driver.call.rsignaling.set_action(rs_action = False)
```

Switches the reduced signaling connection on or off, i.e. activates or deactivates the dedicated (and shared) downlink channels. As a prerequisite for switching on the connection, the cell signal has to be switched on, see method RsCmwWcdmaSig.Source.Cell.State.value.

param rs_action ON | OFF ON: Switch on the reduced signaling connection OFF: Switch off the reduced signaling connection

7.10.2 Pswitched

SCPI Commands

```
CALL:WCDMa:SIGNaling<Instance>:PSWitched:ACTion
```

class Pswitched

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

set_action(*ps_action: RsCmwWcdmaSig.enums.PswitchedAction*) → None

```
# SCPI: CALL:WCDMa:SIGNaling<instance>:PSWitched:ACTion
driver.call.pswitched.set_action(ps_action = enums.PswitchedAction.ACONnect)
```

Controls the PS connection state. As a prerequisite for setup of a test mode connection in the PS domain, a test mode connection must be set up in the CS domain, see method RsCmwWcdmaSig.Call.Cswitched.action.

param ps_action CONNect | DISConnect | HANDover | ACONnect CONNect: initiate the setup of a mobile terminated HSDPA or HSPA test mode connection DISConnect: release the test mode connection HANDover: execute the handover ACONnect: add PS connection to established CS connection (only for test mode connection type RMC + HSPA)

7.10.3 Cswitched

SCPI Commands

```
CALL:WCDMa:SIGNaling<Instance>:CSWitched:ACTion
```

class Cswitched

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

set_action(*cs_action: RsCmwWcdmaSig.enums.CswitchedAction*) → None

```
# SCPI: CALL:WCDMa:SIGNaling<instance>:CSWitched:ACTion
driver.call.cswitched.set_action(cs_action = enums.CswitchedAction.CONNect)
```

Controls the CS connection state. As a prerequisite for connection setup the DL signal has to be switched on, see method RsCmwWcdmaSig.Source.Cell.State.value.

param cs_action CONNect | DISConnect | SSMS | UNRegister | HANDover CONNect: Initiate a CS connection setup DISConnect: Release a CS connection SSMS: Send SMS UNRegister: Unregister the UE completely (CS unregister and PS detach) , i.e. change to state 'On' HANDover: Initiate a handover

7.11 Source

class Source
Source 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.source.clone()
```

Subgroups

7.11.1 Cell

class Cell
Cell 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.source.cell.clone()
```

Subgroups

7.11.1.1 State

SCPI Commands

```
SOURce:WCDMa:SIGNaling<Instance>:CELL:STATe:ALL
SOURce:WCDMa:SIGNaling<Instance>:CELL: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.GeneratorState: OFF | ON | RFHandover OFF: generator switched off ON: generator has been turned on RFHandover: ready to receive a handover from another signaling application
- 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:WCDMa:SIGNaling<instance>:CELL:STATe:ALL
value: AllStruct = driver.source.cell.state.get_all()
```

Returns detailed information about the 'WCDMA Signaling' generator state.

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

get_value() → bool

```
# SCPI: SOURce:WCDMa:SIGNaling<instance>:CELL:STATe
value: bool = driver.source.cell.state.get_value()
```

Turns the generator (the cell) on or off.

return main_state: No help available

set_value(*main_state: bool*) → None

```
# SCPI: SOURce:WCDMa:SIGNaling<instance>:CELL:STATe
driver.source.cell.state.set_value(main_state = False)
```

Turns the generator (the cell) on or off.

param main_state No help available

7.12 Ber

SCPI Commands

```
CALCulate:WCDMa:SIGNaling<Instance>:BER
FETCh:WCDMa:SIGNaling<Instance>:BER
READ:WCDMa:SIGNaling<Instance>:BER
STOP:WCDMa:SIGNaling<Instance>:BER
ABORt:WCDMa:SIGNaling<Instance>:BER
INITiate:WCDMa:SIGNaling<Instance>:BER
```

class Ber

Ber commands group definition. 8 total commands, 1 Sub-groups, 6 group commands

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Ber: enums.ResultStatus2: Bit error rate Range: 0 % to 100 %, Unit: %
- Bler: enums.ResultStatus2: Block error ratio Range: 0 % to 100 %, Unit: %
- Dbler: enums.ResultStatus2: Data block error rate Range: 0 % to 100 %, Unit: %
- Lost_Trans_Blocks: enums.ResultStatus2: Difference between the number of blocks sent and the number of blocks received Range: 0 to total number of blocks sent
- Ult_Fci_Faults: enums.ResultStatus2: Percentage of transport blocks which the UE receiver detected with a wrong transport format, irrespective of the result of the CRC checks Range: 0 % to 100 %, Unit: %
- Fdr: enums.ResultStatus2: False transport format detection ratio; the percentage of transport blocks which passed the UE receiver's CRC check but were detected with a wrong transport format Range: 0 % to 100 %, Unit: %

- Pn_Discontinuity: enums.ResultStatus2: Number of transport blocks that the R&S CMW corrected (i.e. reordered) in the PN resync procedure Range: 0 to total number of blocks sent

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Ber: float: Bit error rate Range: 0 % to 100 %, Unit: %
- Bler: float: Block error ratio Range: 0 % to 100 %, Unit: %
- Dbler: float: Data block error rate Range: 0 % to 100 %, Unit: %
- Lost_Trans_Blocks: int: Difference between the number of blocks sent and the number of blocks received Range: 0 to total number of blocks sent
- Ult_Fci_Faults: float: Percentage of transport blocks which the UE receiver detected with a wrong transport format, irrespective of the result of the CRC checks Range: 0 % to 100 %, Unit: %
- Fdr: float: False transport format detection ratio; the percentage of transport blocks which passed the UE receiver's CRC check but were detected with a wrong transport format Range: 0 % to 100 %, Unit: %
- Pn_Discontinuity: int: Number of transport blocks that the R&S CMW corrected (i.e. reordered) in the PN resync procedure Range: 0 to total number of blocks sent

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:BER
driver.ber.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:BER
driver.ber.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

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

(continues on next page)

(continued from previous page)

```
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

calculate() → CalculateStruct

```
# SCPI: CALCulate:WCDMa:SIGNaling<instance>:BER
value: CalculateStruct = driver.ber.calculate()
```

Returns all results of the signaling BER measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:BER
value: ResultData = driver.ber.fetch()
```

Returns all results of the signaling BER measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:BER
driver.ber.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:BER
driver.ber.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:BER
value: ResultData = driver.ber.read()
```

Returns all results of the signaling BER measurement. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:BER
driver.ber.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:BER
driver.ber.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

Cloning the Group

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

Subgroups

7.12.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:BER:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:BER:STATe
value: enums.ResourceState = driver.ber.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.ber.state.clone()
```

Subgroups

7.12.1.1 All

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:BER: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:WCDMa:SIGNaling<instance>:BER:STATe:ALL
value: FetchStruct = driver.ber.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.

7.13 Throughput

SCPI Commands

```
STOP:WCDMa:SIGNaling<Instance>:THRoughput
ABORt:WCDMa:SIGNaling<Instance>:THRoughput
INITiate:WCDMa:SIGNaling<Instance>:THRoughput
FETCh:WCDMa:SIGNaling<Instance>:THRoughput
READ:WCDMa:SIGNaling<Instance>:THRoughput
```

class Throughput

Throughput commands group definition. 23 total commands, 2 Sub-groups, 5 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Curr_Dl_Pdu: float: Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Avg_Dl_Pdu: float: Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Max_Dl_Pdu: float: Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Min_Dl_Pdu: float: Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Curr_Dl_Sdu: float: Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Avg_Dl_Sdu: float: Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Max_Dl_Sdu: float: Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Min_Dl_Sdu: float: Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Blocks_Dl_Pdu: int: Number of transmitted RLC PDUs Range: 0 to 4E+9
- Curr_Ul_Pdu: float: Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Avg_Ul_Pdu: float: Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Max_Ul_Pdu: float: Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Min_Ul_Pdu: float: Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Curr_Ul_Sdu: float: Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Avg_Ul_Sdu: float: Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Max_Ul_Sdu: float: Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Min_Ul_Sdu: float: Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Blocks_Ul_Pdu: float: Number of received RLC PDUs Range: 0 to 4E+9

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:THRoughput
driver.throughput.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:THRoughput
driver.throughput.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput
value: ResultData = driver.throughput.fetch()
```

Returns all single value throughput results.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:THRoughput
driver.throughput.initiate()
```

(continues on next page)

(continued from previous page)

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

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:THRoughput
driver.throughput.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput
value: ResultData = driver.throughput.read()
```

Returns all single value throughput results.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:THRoughput
driver.throughput.stop()


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

(continues on next page)

(continued from previous page)

```
    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:THRoughput
driver.throughput.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as stop, but waits for the operation to complete before continuing further. Use the RsCmwWcdmaSig.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

7.13.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa: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

7.13.1.1 All

SCPI Commands

```
FETCh:WCDMa: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:WCDMa: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.

7.13.2 Trace

class Trace
Trace commands group definition. 16 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.13.2.1 Downlink

class Downlink
Downlink commands group definition. 8 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.13.2.1.1 Sdu

class Sdu
Sdu 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.throughput.trace.downlink.sdu.clone()
```

Subgroups

7.13.2.1.1.1 Current

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:CURRent
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:CURRent
```

class Current

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:CURRent
value: List[float] = driver.throughput.trace.downlink.sdu.current.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:CURRent
value: List[float] = driver.throughput.trace.downlink.sdu.current.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.13.2.1.1.2 Average

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:AVERage
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:AVERage
```

class Average

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:AVERage
value: List[float] = driver.throughput.trace.downlink.sdu.average.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:AVERage
value: List[float] = driver.throughput.trace.downlink.sdu.average.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.13.2.1.2 Pdu

class Pdu

Pdu 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.throughput.trace.downlink.pdu.clone()
```

Subgroups

7.13.2.1.2.1 Current

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:CURRent
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:CURRent
```

class Current

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
value: List[float] = driver.throughput.trace.downlink.pdu.current.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
value: List[float] = driver.throughput.trace.downlink.pdu.current.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.13.2.1.2.2 Average

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:AVERage
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:AVERage
```

class Average

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
value: List[float] = driver.throughput.trace.downlink.pdu.average.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
value: List[float] = driver.throughput.trace.downlink.pdu.average.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return downlink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.13.2.2 Uplink

class Uplink
Uplink commands group definition. 8 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.13.2.2.1 Sdu

class Sdu
Sdu 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.throughput.trace.uplink.sdu.clone()
```

Subgroups

7.13.2.2.1.1 Current

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:CURRent
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:CURRent
```

class Current
Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:CURRent
value: List[float] = driver.throughput.trace.uplink.sdu.current.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:CURRent
value: List[float] = driver.throughput.trace.uplink.sdu.current.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.13.2.2.1.2 Average

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:AVERage
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:AVERage
```

class Average

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:AVERage
value: List[float] = driver.throughput.trace.uplink.sdu.average.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:AVERage
value: List[float] = driver.throughput.trace.uplink.sdu.average.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_sdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.13.2.2.2 Pdu

class Pdu

Pdu 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.throughput.trace.uplink.pdu.clone()
```

Subgroups

7.13.2.2.2.1 Current

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:CURRent
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:CURRent
```

class Current

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
value: List[float] = driver.throughput.trace.uplink.pdu.current.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
value: List[float] = driver.throughput.trace.uplink.pdu.current.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.13.2.2.2.2 Average

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:AVERage
READ:WCDMa:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:AVERage
```

class Average

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
value: List[float] = driver.throughput.trace.uplink.pdu.average.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
value: List[float] = driver.throughput.trace.uplink.pdu.average.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values N depends on the configured <update interval> and <window size>: N = integer (<window size> / <update interval>)

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

return uplink_pdu: Comma-separated list of N throughput trace values Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.14 Hack

SCPI Commands

```
STOP:WCDMa:SIGNaling<Instance>:HACK
ABORt:WCDMa:SIGNaling<Instance>:HACK
INITiate:WCDMa:SIGNaling<Instance>:HACK
```

class Hack

Hack commands group definition. 39 total commands, 7 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:HACK
driver.hack.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:HACK
driver.hack.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:HACK
driver.hack.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

(continues on next page)

(continued from previous page)

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:HACK
driver.hack.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:HACK
driver.hack.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:HACK
driver.hack.stop_with_opc()


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

(continues on next page)

(continued from previous page)

```
    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

Cloning the Group

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

Subgroups

7.14.1 Trace

class Trace

Trace commands group definition. 22 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.14.1.1 Subframe

class Subframe

Subframe commands group definition. 12 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.14.1.1.1 Carrier

class Carrier
Carrier commands group definition. 12 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.14.1.1.1.1 Tblock

class Tblock
Tblock 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.hack.trace.subframe.carrier.tblock.clone()
```

Subgroups

7.14.1.1.1.2 Minimum

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:TBLock:MINimum
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:TBLock:MINimum
```

class Minimum
Minimum commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:TBLock:MINimum
value: List[int] = driver.hack.trace.subframe.carrier.tblock.minimum.fetch()
```

Returns the trace results per carrier with details on transport block size in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return block: Detected transport block size index Range: 0 to 7

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:TBLock:MINimum
value: List[int] = driver.hack.trace.subframe.carrier.tblock.minimum.read()
```

Returns the trace results per carrier with details on transport block size in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return block: Detected transport block size index Range: 0 to 7

7.14.1.1.1.3 Maximum

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:TBLock:MAXimum
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:TBLock:MAXimum
```

class Maximum

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

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:TBLock:MAXimum
value: List[int] = driver.hack.trace.subframe.carrier.tblock.maximum.fetch()
```

Returns the trace results per carrier with details on transport block size in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return block: Detected transport block size index Range: 0 to 7

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:TBLock:MAXimum
value: List[int] = driver.hack.trace.subframe.carrier.tblock.maximum.read()
```

Returns the trace results per carrier with details on transport block size in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return block: Detected transport block size index Range: 0 to 7

7.14.1.1.1.4 Code

class Code

Code 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.hack.trace.subframe.carrier.code.clone()
```

Subgroups

7.14.1.1.1.5 Minimum

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:CODE:MINimum
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:CODE:MINimum
```

class Minimum

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

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:CODE:MINimum
value: List[int] = driver.hack.trace.subframe.carrier.code.minimum.fetch()
```

Returns the trace results per carrier with details on coding in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return code: Number of detected codes Range: 1 to 15

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:CODE:MINimum
value: List[int] = driver.hack.trace.subframe.carrier.code.minimum.read()
```

Returns the trace results per carrier with details on coding in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return code: Number of detected codes Range: 1 to 15

7.14.1.1.1.6 Maximum

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:CODE:MAXimum
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:CODE:MAXimum
```

class Maximum

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

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:CODE:MAXimum
value: List[int] = driver.hack.trace.subframe.carrier.code.maximum.fetch()
```

Returns the trace results per carrier with details on coding in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return code: Number of detected codes Range: 1 to 15

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:CODE:MAXimum
value: List[int] = driver.hack.trace.subframe.carrier.code.maximum.read()
```

Returns the trace results per carrier with details on coding in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return code: Number of detected codes Range: 1 to 15

7.14.1.1.1.7 Modulation

class Modulation

Modulation 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.hack.trace.subframe.carrier.modulation.clone()
```

Subgroups

7.14.1.1.1.8 Minimum

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:MODulation:MINimum
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:MODulation:MINimum
```

class Minimum

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

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:MODulation:MINimum
value: List[int] = driver.hack.trace.subframe.carrier.modulation.minimum.fetch()
```

Returns the trace results per carrier with details on modulation in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return modulation: QPSK | Q16 | Q64 QPSK, 16-QAM, 64-QAM

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:MODulation:MINimum
value: List[int] = driver.hack.trace.subframe.carrier.modulation.minimum.read()
```

Returns the trace results per carrier with details on modulation in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return modulation: QPSK | Q16 | Q64 QPSK, 16-QAM, 64-QAM

7.14.1.1.1.9 Maximum

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:MODulation:MAXimum
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:SUBFrame:CARRier<Carrier>:MODulation:MAXimum
```

class Maximum

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

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:MODulation:MAXimum
value: List[int] = driver.hack.trace.subframe.carrier.modulation.maximum.fetch()
```

Returns the trace results per carrier with details on modulation in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return modulation: QPSK | Q16 | Q64 QPSK, 16-QAM, 64-QAM

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:SUBFrame:CARRier<carrier>
↪:MODulation:MAXimum
value: List[int] = driver.hack.trace.subframe.carrier.modulation.maximum.read()
```

Returns the trace results per carrier with details on modulation in subframes. Commands query minimum or maximum values. The number of results depends on the configured number of subframes N to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return modulation: QPSK | Q16 | Q64 QPSK, 16-QAM, 64-QAM

7.14.1.2 Throughput

class Throughput
Throughput commands group definition. 8 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.14.1.2.1 Total

class Total
Total 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.hack.trace.throughput.total.clone()
```

Subgroups

7.14.1.2.1.1 Average

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:TOTal:AVERage
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:TOTal:AVERage
```

class Average
Average commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:TOTal:AVERage
value: List[float] = driver.hack.trace.throughput.total.average.fetch()
```

Returns the current overall throughput trace results (sum of all carriers in a multi-carrier scenario) . The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig. Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

return average: Current: n throughput values, from first to last (most recent) measured subframe, one value per 100 measured subframes Average: average of all 'Current' values referenced to the last statistics cycle Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:TOTal:AVERage
value: List[float] = driver.hack.trace.throughput.total.average.read()
```

Returns the current overall throughput trace results (sum of all carriers in a multi-carrier scenario) . The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig. Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

return average: Current: n throughput values, from first to last (most recent) measured subframe, one value per 100 measured subframes Average: average of all 'Current' values referenced to the last statistics cycle Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.14.1.2.1.2 Current

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:TOTal:CURRent
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:TOTal:CURRent
```

class Current

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:TOTal:CURRent
value: List[float] = driver.hack.trace.throughput.total.current.fetch()
```

Returns the current overall throughput trace results (sum of all carriers in a multi-carrier scenario) . The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig. Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

return current: Current: n throughput values, from first to last (most recent) measured subframe, one value per 100 measured subframes Average: average of all 'Current' values referenced to the last statistics cycle Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:TOTal:CURRent
value: List[float] = driver.hack.trace.throughput.total.current.read()
```

Returns the current overall throughput trace results (sum of all carriers in a multi-carrier scenario) . The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig. Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

return current: Current: n throughput values, from first to last (most recent) measured subframe, one value per 100 measured subframes Average: average of all 'Current' values referenced to the last statistics cycle Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.14.1.2.2 Carrier

class Carrier

Carrier 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.hack.trace.throughput.carrier.clone()
```

Subgroups

7.14.1.2.2.1 Average

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:CARRier<Carrier>:AVERage
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:CARRier<Carrier>:AVERage
```

class Average

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:CARRier<carrier>
↪:AVERage
value: List[float] = driver.hack.trace.throughput.carrier.average.fetch()
```

Returns the current throughput trace results per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return average: No help available

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:CARRier<carrier>
↪:AVERage
value: List[float] = driver.hack.trace.throughput.carrier.average.read()
```

Returns the current throughput trace results per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return average: No help available

7.14.1.2.2.2 Current

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:CARRier<Carrier>:CURRent
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:THRoughput:CARRier<Carrier>:CURRent
```

class Current

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:CARRier<carrier>
↪:CURRent
value: List[float] = driver.hack.trace.throughput.carrier.current.fetch()
```

Returns the current throughput trace results per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return current: No help available

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:THRoughput:CARRier<carrier>
↪:CURRent
value: List[float] = driver.hack.trace.throughput.carrier.current.read()
```

Returns the current throughput trace results per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames. For each 100 subframes, one result is returned. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return current: No help available

7.14.1.3 Mcqi

class Mcqi
Mcqi 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.hack.trace.mcqi.clone()
```

Subgroups

7.14.1.3.1 Carrier

class Carrier
Carrier 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.hack.trace.mcqi.carrier.clone()
```

Subgroups

7.14.1.3.1.1 Current

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRACe:MCQI:CARRier<Carrier>:CURRent
READ:WCDMa:SIGNaling<Instance>:HACK:TRACe:MCQI:CARRier<Carrier>:CURRent
```

class Current
Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRACe:MCQI:CARRier<carrier>:CURRent
value: List[int] = driver.hack.trace.mcqi.carrier.current.fetch()
```

Returns the current median CQI trace results. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames. For each 100 subframes, one result is returned.

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

Global Repeated Capabilities: repcap.Carrier

return current: N median CQI values, from first to last measured subframe, one value per 100 measured subframes Range: 0 to 31

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRACe:MCQI:CARRier<carrier>:CURRent
value: List[int] = driver.hack.trace.mcqi.carrier.current.read()
```

Returns the current median CQI trace results. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Hack.msFrames. For each 100 subframes, one result is returned.

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

Global Repeated Capabilities: repcap.Carrier

return current: N median CQI values, from first to last measured subframe, one value per 100 measured subframes Range: 0 to 31

7.14.2 Mcqi

class Mcqi

Mcqi 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.hack.mcqi.clone()
```

Subgroups

7.14.2.1 Carrier

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:MCQI:CARRier<Carrier>
READ:WCDMa:SIGNaling<Instance>:HACK:MCQI:CARRier<Carrier>
```

class Carrier

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

fetch() → int

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:MCQI:CARRier<carrier>
value: int = driver.hack.mcqi.carrier.fetch()
```

Return the median CQI result per carrier, see 'Median CQI'.

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

Global Repeated Capabilities: repcap.Carrier

return median_cqi: Range: 0 to 31

read() → int

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:MCQI:CARRier<carrier>
value: int = driver.hack.mcqi.carrier.read()
```

Return the median CQI result per carrier, see 'Median CQI'.

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

Global Repeated Capabilities: repcap.Carrier

return median_cqi: Range: 0 to 31

7.14.3 MsFrames

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:MSFRames
READ:WCDMa:SIGNaling<Instance>:HACK:MSFRames
```

class MsFrames

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

fetch() → int

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:MSFRames
value: int = driver.hack.msFrames.fetch()
```

Return the total number of already measured HSDPA subframes.

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

return meas_sub_frames: Range: 0 to 2E+9

read() → int

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:MSFRames
value: int = driver.hack.msFrames.read()
```

Return the total number of already measured HSDPA subframes.

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

return meas_sub_frames: Range: 0 to 2E+9

7.14.4 Bler

class Bler

Bler 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.hack.bler.clone()
```

Subgroups

7.14.4.1 Carrier

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:BLER:CARRier<Carrier>
READ:WCDMa:SIGNaling<Instance>:HACK:BLER:CARRier<Carrier>
```

class Carrier

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

fetch() → float

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:BLER:CARRier<carrier>
value: float = driver.hack.bler.carrier.fetch()
```

Return the BLER result per carrier, see 'DL BLER'.

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

Global Repeated Capabilities: repcap.Carrier

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

read() → float

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:BLER:CARRier<carrier>
value: float = driver.hack.bler.carrier.read()
```

Return the BLER result per carrier, see 'DL BLER'.

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

Global Repeated Capabilities: repcap.Carrier

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

7.14.5 Throughput

class Throughput

Throughput commands group definition. 4 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.14.5.1 Carrier

class Carrier
Carrier 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.hack.throughput.carrier.clone()
```

Subgroups

7.14.5.1.1 Relative

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:THRoughput:CARRier<Carrier>:RELative
READ:WCDMa:SIGNaling<Instance>:HACK:THRoughput:CARRier<Carrier>:RELative
```

class Relative
Relative commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ResultData
Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Rel_Current: float: Range: 0 % to 100 %, Unit: %
- Rel_Maximum: float: Range: 0 % to 100 %, Unit: %
- Rel_Minimum: float: Range: 0 % to 100 %, Unit: %
- Rel_Scheduled: float: Range: 0 % to 100 %, Unit: %
- Rel_Average: float: Range: 0 % to 100 %, Unit: %

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:THRoughput:CARRier<carrier>
↪:RELative
value: ResultData = driver.hack.throughput.carrier.relative.fetch()
```

Return the throughput results as percentage of the 'Max. possible Throughput'. The current, maximum, minimum, scheduled and average values are returned, see 'Throughput'.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:THRoughput:CARRier<carrier>:RELative
value: ResultData = driver.hack.throughput.carrier.relative.read()
```

Return the throughput results as percentage of the 'Max. possible Throughput'. The current, maximum, minimum, scheduled and average values are returned, see 'Throughput'.

Global Repeated Capabilities: repcap.Carrier

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

7.14.5.1.2 Absolute

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:THRoughput:CARRier<Carrier>:ABSolute
READ:WCDMa:SIGNaling<Instance>:HACK:THRoughput:CARRier<Carrier>:ABSolute
```

class Absolute

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Abs_Current: float: Current throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Abs_Maximum: float: Maximum throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Abs_Minimum: float: Minimum throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Abs_Scheduled: float: Scheduled throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Max_Possible: float: Maximum possible throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Abs_Total_Current: float: Current throughput - sum of all carriers Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Total_Max_Pos: float: Maximum possible throughput - sum of all carriers Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Abs_Total_Average: float: Average throughput calculated from a sum of all carriers Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Abs_Average: float: Average throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:THRoughput:CARRier<carrier>
↪:ABSolute
value: ResultData = driver.hack.throughput.carrier.absolute.fetch()
```

Return the throughput results as absolute values. The current, maximum, minimum, scheduled and average values are returned, see 'Throughput'. In addition to the measured values, the theoretical maximum possible throughput is returned, see 'Max. possible Throughput'.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:THRoughput:CARRier<carrier>:ABSolute
value: ResultData = driver.hack.throughput.carrier.absolute.read()
```

Return the throughput results as absolute values. The current, maximum, minimum, scheduled and average values are returned, see 'Throughput'. In addition to the measured values, the theoretical maximum possible throughput is returned, see 'Max. possible Throughput'.

Global Repeated Capabilities: repcap.Carrier

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

7.14.6 Transmission

class Transmission

Transmission 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.hack.transmission.clone()
```

Subgroups

7.14.6.1 Carrier

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:TRANsmission:CARRier<Carrier>
READ:WCDMa:SIGNaling<Instance>:HACK:TRANsmission:CARRier<Carrier>
```

class Carrier

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Transmitted: List[float]: No parameter help available
- Ack: List[float]: No parameter help available
- Nack: List[float]: No parameter help available
- Dtx: List[float]: No parameter help available

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:TRANsmission:CARRier<carrier>
value: ResultData = driver.hack.transmission.carrier.fetch()
```

Return all results of the 'Transmissions' table row by row, see 'Transmissions'.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HACK:TRANsmission:CARRier<carrier>
value: ResultData = driver.hack.transmission.carrier.read()
```

Return all results of the 'Transmissions' table row by row, see 'Transmissions'.

Global Repeated Capabilities: repcap.Carrier

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

7.14.7 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HACK:STATe
value: enums.ResourceState = driver.hack.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.hack.state.clone()
```

Subgroups

7.14.7.1 All

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HACK: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:WCDMa:SIGNaling<instance>:HACK:STATe:ALL
value: FetchStruct = driver.hack.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.

7.15 Hcqi

SCPI Commands

```
STOP:WCDMa:SIGNaling<Instance>:HCQI
ABORt:WCDMa:SIGNaling<Instance>:HCQI
INITiate:WCDMa:SIGNaling<Instance>:HCQI
```

class Hcqi

Hcqi commands group definition. 16 total commands, 4 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:HCQI
driver.hcqi.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:HCQI
driver.hcqi.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:HCQI
driver.hcqi.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

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

(continues on next page)

(continued from previous page)

```
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:HCQI
driver.hcqi.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:HCQI
driver.hcqi.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:HCQI
driver.hcqi.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

Cloning the Group

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

Subgroups

7.15.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HCQI:STATe
value: enums.ResourceState = driver.hcqi.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.hcqi.state.clone()
```

Subgroups

7.15.1.1 All

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI: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:WCDMa:SIGNaling<instance>:HCQI:STATe:ALL
value: FetchStruct = driver.hcqi.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.

7.15.2 Rstate

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI:RSTate
```

class Rstate

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

fetch() → RsCmwWcdmaSig.enums.ResultState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HCQI:RSTate
value: enums.ResultState = driver.hcqi.rstate.fetch()
```

Queries the result of the entire HSDPA CQI measurement including all stages.

return result_state: FAIL | PASS | RUN Measurement failed, passed, running.

7.15.3 Carrier

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>
READ:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>
```

class Carrier

Carrier commands group definition. 8 total commands, 3 Sub-groups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Median_Cqi: int: Middle of the CQI distribution reported in the first measurement stage Range: 0 to 30
- Meas_Sub_Frames: int: Total number of measured HSDPA subframes in stage one Range: 0 to 1E+6
- Cqiin_Range: float: Percentage of the CQI values reported within the interval [median CQI - 2, median CQI + 2] Range: 0 % to 100 %, Unit: %

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>
value: ResultData = driver.hcqi.carrier.fetch()
```

Returns the results of the first stage of HSDPA CQI measurement per carrier.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>
value: ResultData = driver.hcqi.carrier.read()
```

Returns the results of the first stage of HSDPA CQI measurement per carrier.

Global Repeated Capabilities: repcap.Carrier

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.hcqi.carrier.clone()
```

Subgroups

7.15.3.1 Bler

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>:BLER
READ:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>:BLER
```

class Bler

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Median_Cqim_1: float: Block error rate measured at median CQI - 1 in the third stage of measurement (AWGN test case only) Range: 0 % to 100 %, Unit: %
- Median_Cqi: float: Block error rate measured at median CQI in the second stage of measurement (AWGN and fading test cases) Range: 0 % to 100 %, Unit: %
- Median_Cqip_2: float: Block error rate measured at median CQI + 2 in the third stage of measurement (AWGN test case only) Range: 0 % to 100 %, Unit: %
- Median_Cqip_3: float: Block error rate measured at median CQI + 3 in the second stage of measurement (Fading test case only) Range: 0 % to 100 %, Unit: %

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>:BLER
value: ResultData = driver.hcqi.carrier.bler.fetch()
```

Returns the BLER results of the second and third stage of HSDPA CQI measurement. As indicated in the parameter descriptions below, each test case provides valid results for a subset of the parameters only. For the other parameters NCAP is returned.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>:BLER
value: ResultData = driver.hcqi.carrier.bler.read()
```

Returns the BLER results of the second and third stage of HSDPA CQI measurement. As indicated in the parameter descriptions below, each test case provides valid results for a subset of the parameters only. For the other parameters NCAP is returned.

Global Repeated Capabilities: repcap.Carrier

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

7.15.3.2 Dtx

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>:DTX
READ:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>:DTX
```

class Dtx

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Median_Cqim_1: float: Percentage of DTX responses measured at median CQI - 1 in the third stage of measurement (AWGN test case only) Additional parameter: On|Off enables/disables the DTX statistics Range: 0 % to 100 %, Unit: %
- Median_Cqi: float: Percentage of DTX responses measured at median CQI in the second stage of measurement (AWGN and fading test cases) Additional parameter: On|Off enables/disables the DTX statistics Range: 0 % to 100 %, Unit: %
- Median_Cqip_2: float: Percentage of DTX responses measured at median CQI + 2 in the third stage of measurement (AWGN test case only) Additional parameter: On|Off enables/disables the DTX statistics Range: 0 % to 100 %, Unit: %
- Median_Cqip_3: float: Percentage of DTX responses measured at median CQI + 3 in the second stage of measurement (Fading test case only) Additional parameter: On|Off enables/disables the DTX statistics Range: 0 % to 100 %, Unit: %

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>:DTX
value: ResultData = driver.hcqi.carrier.dtx.fetch()
```

Returns the DTX results of the second and third stage of HSDPA CQI measurement. As indicated in the parameter descriptions below, each test case provides valid results for a subset of the parameters only. For the other parameters NCAP is returned.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>:DTX
value: ResultData = driver.hcqi.carrier.dtx.read()
```

Returns the DTX results of the second and third stage of HSDPA CQI measurement. As indicated in the parameter descriptions below, each test case provides valid results for a subset of the parameters only. For the other parameters NCAP is returned.

Global Repeated Capabilities: repcap.Carrier

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

7.15.3.3 MsFrames

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>:MSFRames
READ:WCDMa:SIGNaling<Instance>:HCQI:CARRier<Carrier>:MSFRames
```

class MsFrames

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Median_Cqim_1: int: The number of subframes with ACK and NACK responses measured at median CQI - 1 in the third stage of measurement (AWGN test case only) Range: 0 to 1E+6
- Median_Cqi: int: The number of subframes with ACK and NACK responses measured at median CQI in the second stage of measurement (AWGN and fading test cases) Range: 0 to 1E+6
- Median_Cqip_2: int: The number of subframes with ACK and NACK responses measured at median CQI + 2 in the third stage of measurement (AWGN test case only) Range: 0 to 1E+6
- Median_Cqip_3: int: The number of subframes with ACK and NACK responses measured at median CQI + 3 in the second stage of measurement (Fading test case only) Range: 0 to 1E+6

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>:MSFRames
value: ResultData = driver.hcqi.carrier.msFrames.fetch()
```

Returns the number of subframes measured during the second and third stage of HSDPA CQI measurement to calculate BLER and DTX. As indicated in the parameter descriptions below, each test case provides valid results for a subset of the parameters only. For the other parameters NCAP is returned.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HCQI:CARRier<carrier>:MSFRames
value: ResultData = driver.hcqi.carrier.msFrames.read()
```

Returns the number of subframes measured during the second and third stage of HSDPA CQI measurement to calculate BLER and DTX. As indicated in the parameter descriptions below, each test case provides valid results for a subset of the parameters only. For the other parameters NCAP is returned.

Global Repeated Capabilities: repcap.Carrier

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

7.15.4 Trace

class Trace
Trace 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.hcqi.trace.clone()
```

Subgroups

7.15.4.1 Carrier

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:HCQI:TRACe:CARRier<Carrier>
READ:WCDMa:SIGNaling<Instance>:HCQI:TRACe:CARRier<Carrier>
```

class Carrier
Carrier commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:HCQI:TRACe:CARRier<carrier>
value: List[float] = driver.hcqi.trace.carrier.fetch()
```

Returns the CQI distribution results in percentage per carrier. For each CQI value one result is returned: <Reliability>, <HistCQI>0, ..., <HistCQI>31

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

Global Repeated Capabilities: repcap.Carrier

return hist_cqi: Histogram CQI: percentage of the reported CQI value 0 to 30 per measurement cycle The position 31 indicates the percentage of DTX subframes. Range: 0 % to 100 %, Unit: %

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:HCQI:TRACe:CARRier<carrier>
value: List[float] = driver.hcqi.trace.carrier.read()
```

Returns the CQI distribution results in percentage per carrier. For each CQI value one result is returned: <Reliability>, <HistCQI>0, ..., <HistCQI>31

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

Global Repeated Capabilities: repcap.Carrier

return hist_cqi: Histogram CQI: percentage of the reported CQI value 0 to 30 per measurement cycle The position 31 indicates the percentage of DTX subframes. Range: 0 % to 100 %, Unit: %

7.16 UplinkLogging

SCPI Commands

```
STOP:WCDMa:SIGNaling<Instance>:ULLogging
ABORt:WCDMa:SIGNaling<Instance>:ULLogging
INITiate:WCDMa:SIGNaling<Instance>:ULLogging
```

class UplinkLogging

UplinkLogging commands group definition. 27 total commands, 7 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:ULLogging
driver.uplinkLogging.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:ULLogging
driver.uplinkLogging.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:ULLogging
driver.uplinkLogging.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:ULLogging
driver.uplinkLogging.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:ULLogging
driver.uplinkLogging.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
```

(continues on next page)

(continued from previous page)

```
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:ULLogging
driver.uplinkLogging.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

Cloning the Group

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

Subgroups

7.16.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:STATe
value: enums.ResourceState = driver.uplinkLogging.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.uplinkLogging.state.clone()
```

Subgroups

7.16.1.1 All

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging: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:WCDMa:SIGNaling<instance>:ULLogging:STATe:ALL
value: FetchStruct = driver.uplinkLogging.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.

7.16.2 Sfn

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:SFN
READ:WCDMa:SIGNaling<Instance>:ULLogging:SFN
```

class Sfn

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

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:SFN
value: List[int] = driver.uplinkLogging.sfn.fetch()
```

Return results of the UL logging measurement on the UL HS-DPCCH/E-DPCCH/DPCCH. The results are returned per measured subframe: <Reliability>, <SFN>subframe1, <SFN>subframe2, ..., <SFN>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

return sfn: System frame number corresponds to the subframe number for which the UL logging information is displayed (set to modulo 4095) Range: 0 to 4095

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:SFN
value: List[int] = driver.uplinkLogging.sfn.read()
```

Return results of the UL logging measurement on the UL HS-DPCCH/E-DPCCH/DPCCH. The results are returned per measured subframe: <Reliability>, <SFN>subframe1, <SFN>subframe2, ..., <SFN>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

return sfn: System frame number corresponds to the subframe number for which the UL logging information is displayed (set to modulo 4095) Range: 0 to 4095

7.16.3 Slot

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:SLOT
READ:WCDMa:SIGNaling<Instance>:ULLogging:SLOT
```

class Slot

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

fetch() → List[int]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:SLOT
value: List[int] = driver.uplinkLogging.slot.fetch()
```

Return results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned per measured subframe: <Reliability>, <Slot>subframe1, <Slot>subframe2, ..., <Slot>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

return slot: First slot number of the received UL HS-DPCCH/E-DPCCH/DPCCH subframe; see 'UL Logging Measurement' Range: 0 | 3 | 6 | 9 | 12

read() → List[int]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:SLOT
value: List[int] = driver.uplinkLogging.slot.read()
```

Return results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned per measured subframe: <Reliability>, <Slot>subframe1, <Slot>subframe2, ..., <Slot>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

return slot: First slot number of the received UL HS-DPCCH/E-DPCCH/DPCCH subframe; see 'UL Logging Measurement' Range: 0 | 3 | 6 | 9 | 12

7.16.4 Carrier

class Carrier

Carrier commands group definition. 12 total commands, 6 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.16.4.1 Etfci

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:ETFCi
READ:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:ETFCi
```

class Etfci

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

fetch() → List[RsCmwWcdmaSig.enums.Etfci]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:ETFCi
value: List[enums.Etfci] = driver.uplinkLogging.carrier.etfci.fetch()
```

Return results of the UL logging measurement on the E-DPCCH. The results are returned per measured subframe: <Reliability>, <ETFCI>subframe1, <ETFCI>subframe2, ..., <ETFCI>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return etfci: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 See also Table '2ms TTI E-DCH transport block size' DTX: no answer received from the UE 0 to 127: indicates the transport block size on the E-DPDCH

read() → List[RsCmwWcdmaSig.enums.Etfci]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:ETFCi
value: List[enums.Etfci] = driver.uplinkLogging.carrier.etfci.read()
```

Return results of the UL logging measurement on the E-DPCCH. The results are returned per measured subframe: <Reliability>, <ETFCI>subframe1, <ETFCI>subframe2, ..., <ETFCI>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return etfci: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 See also Table '2ms TTI E-DCH transport block size' DTX: no answer received from the UE 0 to 127: indicates the transport block size on the E-DPDCH

7.16.4.2 Rsn

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:RSN
READ:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:RSN
```

class Rsn

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

fetch() → List[RsCmwWcdmaSig.enums.RetransmisionSeqNr]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:RSN
value: List[enums.RetransmisionSeqNr] = driver.uplinkLogging.carrier.rsn.fetch()
```

Return results of the UL logging measurement on the E-DPCCH. The results are returned per measured subframe: <Reliability>, <RSN>subframe1, <RSN>subframe2, ..., <RSN>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return rsn: DTX | 0 | 1 | 2 | 3 Retransmission sequence number: DTX: no answer received from the UE 0: new transmission 1: first retransmission 2: second retransmission 3: higher than second retransmission

read() → List[RsCmwWcdmaSig.enums.RetransmisionSeqNr]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:RSN
value: List[enums.RetransmisionSeqNr] = driver.uplinkLogging.carrier.rsn.read()
```

Return results of the UL logging measurement on the E-DPCCH. The results are returned per measured subframe: <Reliability>, <RSN>subframe1, <RSN>subframe2, ..., <RSN>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return rsn: DTX | 0 | 1 | 2 | 3 Retransmission sequence number: DTX: no answer received from the UE 0: new transmission 1: first retransmission 2: second retransmission 3: higher than second retransmission

7.16.4.3 Hbit

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:HBIT
READ:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:HBIT
```

class Hbit

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

fetch() → List[RsCmwWcdmaSig.enums.HappyBit]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:HBIT
value: List[enums.HappyBit] = driver.uplinkLogging.carrier.hbit.fetch()
```

Return results of the UL logging measurement on the E-DPCCH. The results are returned per measured subframe: <Reliability>, <HappyBit>subframe1, <HappyBit>subframe2, ..., <HappyBit>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return happy_bit: HAPPy | UNHappy | DTX HAPPy: UE is satisfied with the granted data rate UNHappy: UE is not transmitting at maximum power and cannot empty its transmit buffer with the current serving grant within a certain time period DTX: no answer received from the UE

read() → List[RsCmwWcdmaSig.enums.HappyBit]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:HBIT
value: List[enums.HappyBit] = driver.uplinkLogging.carrier.hbit.read()
```

Return results of the UL logging measurement on the E-DPCCH. The results are returned per measured subframe: <Reliability>, <HappyBit>subframe1, <HappyBit>subframe2, ..., <HappyBit>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return happy_bit: HAPPy | UNHappy | DTX HAPPy: UE is satisfied with the granted data rate UNHappy: UE is not transmitting at maximum power and cannot empty its transmit buffer with the current serving grant within a certain time period DTX: no answer received from the UE

7.16.4.4 Dpcch

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:DPCCh
READ:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:DPCCh
```

class Dpcch

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Dpcch_1: List[bool]: OFF | ON Queries the status of DPCCH read out from the first slot
- Dpcch_2: List[bool]: OFF | ON Queries the status of DPCCH read out from the second slot
- Dpcch_3: List[bool]: OFF | ON Queries the status of DPCCH read out from the third slot

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:DPCCh
value: ResultData = driver.uplinkLogging.carrier.dpcch.fetch()
```

Return results of the UL logging measurement on the DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<DPCCH1>, <DPCCH2>, <DPCCH3>}subframe1, {...}subframe 2, ..., {...}subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:DPCCh
value: ResultData = driver.uplinkLogging.carrier.dpcch.read()
```

Return results of the UL logging measurement on the DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<DPCCH1>, <DPCCH2>, <DPCCH3>}subframe1, {...}subframe 2, ..., {...}subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

Global Repeated Capabilities: repcap.Carrier

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

7.16.4.5 Anack

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:ANACk
READ:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:ANACk
```

class Anack

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

fetch() → List[RsCmwWcdmaSig.enums.AckNack]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:ANACk
value: List[enums.AckNack] = driver.uplinkLogging.carrier.anack.fetch()
```

Return results of the UL logging measurement on the UL HS-DPCCH. The results are returned per measured subframe: <Reliability>, <ACKNACK>subframe1, <ACKNACK>subframe2, ..., <ACKNACK>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return ack_nack: DTX | ACK | NACK HARQ-ACK: DTX: no answer received from the UE ACK: successful CRC check of a received transmission packet NACK: failed CRC check of a received transmission packet

read() → List[RsCmwWcdmaSig.enums.AckNack]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:ANACk
value: List[enums.AckNack] = driver.uplinkLogging.carrier.anack.read()
```

Return results of the UL logging measurement on the UL HS-DPCCH. The results are returned per measured subframe: <Reliability>, <ACKNACK>subframe1, <ACKNACK>subframe2, ..., <ACKNACK>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return ack_nack: DTX | ACK | NACK HARQ-ACK: DTX: no answer received from the UE ACK: successful CRC check of a received transmission packet NACK: failed CRC check of a received transmission packet

7.16.4.6 Cqi

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:CQI
READ:WCDMa:SIGNaling<Instance>:ULLogging:CARRier<Carrier>:CQI
```

class Cqi

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

fetch() → List[RsCmwWcdmaSig.enums.Cqi]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:CQI
value: List[enums.Cqi] = driver.uplinkLogging.carrier.cqi.fetch()
```

Return results of the UL logging measurement on the HS-DPCCH. The results are returned per measured subframe: <Reliability>, <CQI>subframe1, <CQI>subframe2, ..., <CQI>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return cqi: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 DTX: no answer received from the UE 0 to 30: reported channel quality indicator, 30 means the best quality

read() → List[RsCmwWcdmaSig.enums.Cqi]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:CARRier<carrier>:CQI
value: List[enums.Cqi] = driver.uplinkLogging.carrier.cqi.read()
```

Return results of the UL logging measurement on the HS-DPCCH. The results are returned per measured subframe: <Reliability>, <CQI>subframe1, <CQI>subframe2, ..., <CQI>subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames.

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

Global Repeated Capabilities: repcap.Carrier

return cqi: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 DTX: no answer received from the UE 0 to 30: reported channel quality indicator, 30 means the best quality

7.16.5 Scell

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:SCELl
READ:WCDMa:SIGNaling<Instance>:ULLogging:SCELl
```

class Scell

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Sfn: List[int]: System frame number corresponds to the subframe number for which the UL HS-DPCCH/E-DPCCH/DPCCH information is displayed (set to modulo 4095) Range: 0 to 4095
- Slot: List[int]: First slot number of the received UL HS-DPCCH/E-DPCCH/DPCCH subframe; see 'UL Logging Measurement' Range: 0 | 3 | 6 | 9 | 12
- Etfci: List[enums.Etfci]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 See also Table '2ms TTI E-DCH transport block size' DTX: no answer received from the UE 0 to 127: indicates the transport block size on the E-DPDCH
- Rsn: List[enums.RetransmisionSeqNr]: No parameter help available
- Happy_Bit: List[enums.HappyBit]: HAPPy | UNHappy | DTX HAPPy: UE is satisfied with the granted data rate UNHappy: UE is not transmitting at maximum power and cannot empty its transmit buffer with the current serving grant within a certain time period DTX: no answer received from the UE
- Dpcch_1: List[bool]: No parameter help available
- Dpcch_2: List[bool]: No parameter help available
- Dpcch_3: List[bool]: No parameter help available
- Ack_Nack: List[enums.AckNack]: No parameter help available
- Cqi: List[enums.Cqi]: No parameter help available

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging[:SCELl]
value: ResultData = driver.uplinkLogging.scell.fetch()
```

Return all results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<SFN>, <Slot>, <ETFCI>, <RSN>, <HappyBit>, <DPCCH1>, <DPCCH2>, <DPCCH3>, <ACKNACK>, <CQI>}subframe 1, {...}subframe 2, ..., {...}subframe n The number of subframes n is configured via method RsCmwWcdmaSig. Configure.UplinkLogging.msFrames. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging[:SCELl]
value: ResultData = driver.uplinkLogging.scell.read()
```

Return all results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<SFN>, <Slot>, <ETFCI>, <RSN>, <HappyBit>, <DPCCH1>, <DPCCH2>, <DPCCH3>, <ACKNACK>, <CQI>}subframe 1, {...}subframe 2, ..., {...}subframe n The number of subframes n is configured via method RsCmwWcdmaSig. Configure.UplinkLogging.msFrames. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

7.16.6 Dcarrier

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:DCARrier
READ:WCDMa:SIGNaling<Instance>:ULLogging:DCARrier
```

class Dcarrier

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Sfn: List[int]: System frame number corresponds to the subframe number for which the UL HS-DPCCH/E-DPCCH/DPCCH information is displayed (set to modulo 4095) Range: 0 to 4095
- Slot: List[int]: First slot number of the received UL HS-DPCCH/E-DPCCH/DPCCH subframe; see 'UL Logging Measurement' Range: 0 | 3 | 6 | 9 | 12
- Etfci: List[enums.Etfci]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 See also Table '2ms TTI E-DCH transport block size' DTX: no answer received from the UE 0 to 127: indicates the transport block size on the E-DPDCH
- Rsn: List[enums.RetransmisionSeqNr]: DTX | 0 | 1 | 2 | 3 Retransmission sequence number DTX: no answer received from the UE 0: new transmission 1: first retransmission 2: second retransmission 3: higher than second retransmission
- Happy_Bit: List[enums.HappyBit]: HAPPy | UNHappy | DTX HAPPy: UE is satisfied with the granted data rate UNHappy: UE is not transmitting at maximum power and cannot empty its transmit buffer with the current serving grant within a certain time period DTX: no answer received from the UE
- Dpcch_1: List[bool]: OFF | ON Queries the status of DPCCH read out from the first slot
- Dpcch_2: List[bool]: OFF | ON Queries the status of DPCCH read out from the second slot

- Dpcch_3: List[bool]: OFF | ON Queries the status of DPCCH read out from the third slot
- Ack_Nack_1: List[enums.AckNack]: DTX | ACK | NACK HARQ ACK: UE response (by dual carrier - carrier one) DTX: no answer received from the UE ACK: successful CRC check of a received transmission packet NACK: failed CRC check of a received transmission packet
- Cqi_1: List[enums.Cqi]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 UE response (by dual carrier - carrier one) ; 30 means the best quality DTX: no answer received from the UE 0 to 30: reported channel quality indicator
- Ack_Nack_2: List[enums.AckNack]: ACK | NACK | DTX HARQ ACK: UE response (by dual carrier - carrier two) ACK: successful CRC check of a received transmission packet NACK: failed CRC check of a received transmission packet DTX: no answer received from the UE
- Cqi_2: List[enums.Cqi]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 UE response (by dual carrier - carrier two) ; 30 means the best quality DTX: no answer received from the UE 0 to 30: reported channel quality indicator

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:DCARrier
value: ResultData = driver.uplinkLogging.dcarrier.fetch()
```

Return all results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<SFN>, <Slot>, <ETFCI>, <RSN>, <HappyBit>, <DPCCH1>, <DPCCH2>, <DPCCH3>, <ACKNACK1>, <CQI1>, <ACK-NACK2>, <CQI2>}subframe 1, {...}subframe 2, ..., {...}subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:DCARrier
value: ResultData = driver.uplinkLogging.dcarrier.read()
```

Return all results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<SFN>, <Slot>, <ETFCI>, <RSN>, <HappyBit>, <DPCCH1>, <DPCCH2>, <DPCCH3>, <ACKNACK1>, <CQI1>, <ACK-NACK2>, <CQI2>}subframe 1, {...}subframe 2, ..., {...}subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

7.16.7 Dchspa

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ULLogging:DCHSpa
READ:WCDMa:SIGNaling<Instance>:ULLogging:DCHSpa
```

class Dchspa

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Sfn: List[int]: System frame number corresponds to the subframe number for which the UL HS-DPCCH/E-DPCCH/DPCCH information is displayed (set to modulo 4095) Range: 0 to 4095
- Slot: List[int]: First slot number of the received UL HS-DPCCH/E-DPCCH/DPCCH subframe; see 'UL Logging Measurement' Range: 0 | 3 | 6 | 9 | 12
- Etfci_1: List[enums.Etfci]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 (Information related to dual carrier HSPA - cell one) See also Table '2ms TTI E-DCH transport block size' DTX: no answer received from the UE 0 to 127: indicates the transport block size on the E-DPDCH
- Rsn_1: List[enums.RetransmisionSeqNr]: DTX | 0 | 1 | 2 | 3 (Information related to dual carrier HSPA - cell one) Retransmission sequence number DTX: no answer received from the UE 0: new transmission 1: first retransmission 2: second retransmission 3: higher than second retransmission
- Happy_Bit_1: List[enums.HappyBit]: HAPPy | UNHappy | DTX (Information related to dual carrier HSPA - cell one) HAPPy: UE is satisfied with the granted data rate UNHappy: UE is not transmitting at maximum power and cannot empty its transmit buffer with the current serving grant within a certain time period DTX: no answer received from the UE
- Dpcch_1_C_1: List[bool]: OFF | ON (Information related to dual carrier HSPA - cell one) Queries the status of DPCCH read out from the first slot
- Dpcch_2_C_1: List[bool]: OFF | ON (Information related to dual carrier HSPA - cell one) Queries the status of DPCCH read out from the second slot
- Dpcch_3_C_1: List[bool]: OFF | ON (Information related to dual carrier HSPA - cell one) Queries the status of DPCCH read out from the third slot
- Etfci_2: List[enums.Etfci]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 (Information related to dual carrier HSPA - cell two) See also Table '2ms TTI E-DCH transport block size' DTX: no answer received from the UE 0 to 127: indicates the transport block size on the E-DPDCH
- Rsn_2: List[enums.RetransmisionSeqNr]: DTX | 0 | 1 | 2 | 3 (Information related to dual carrier HSPA - cell two) Retransmission sequence number DTX: no answer received from the UE 0: new transmission

1: first retransmission 2: second retransmission 3: higher than second retransmission

- Happy_Bit_2: List[enums.HappyBit]: HAPPy | UNHappy | DTX (Information related to dual carrier HSPA - cell two) HAPPy: UE is satisfied with the granted data rate UNHappy: UE is not transmitting at maximum power and cannot empty its transmit buffer with the current serving grant within a certain time period DTX: no answer received from the UE
- Dpcch_1_C_2: List[bool]: OFF | ON (Information related to dual carrier HSPA - cell two) Queries the status of DPCCH read out from the first slot
- Dpcch_2_C_2: List[bool]: OFF | ON (Information related to dual carrier HSPA - cell two) Queries the status of DPCCH read out from the second slot
- Dpcch_3_C_2: List[bool]: OFF | ON (Information related to dual carrier HSPA - cell two) Queries the status of DPCCH read out from the third slot
- Ack_Nack_1: List[enums.AckNack]: DTX | ACK | NACK HARQ ACK: UE response (Information related to dual carrier HSPA - cell one) DTX: no answer received from the UE ACK: successful CRC check of a received transmission packet NACK: failed CRC check of a received transmission packet
- Cqi_1: List[enums.Cqi]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 UE response; 30 means the best quality (Information related to dual carrier HSPA - cell one) DTX: no answer received from the UE 0 to 30: reported channel quality indicator
- Ack_Nack_2: List[enums.AckNack]: DTX | ACK | NACK HARQ ACK: UE response (Information related to dual carrier HSPA - cell two) ACK: successful CRC check of a received transmission packet NACK: failed CRC check of a received transmission packet DTX: no answer received from the UE
- Cqi_2: List[enums.Cqi]: DTX | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 UE response; 30 means the best quality (Information related to dual carrier HSPA - cell two) DTX: no answer received from the UE 0 to 30: reported channel quality indicator

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ULLogging:DCHSpa
value: ResultData = driver.uplinkLogging.dchspa.fetch()
```

Return all results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<SFN>, <Slot>, <ETFCI>, <RSN>, <HappyBit>, <DPCCH1>, <DPCCH2>, <DPCCH3>, <ACKNACK1>, <CQI1>, <ACK-NACK2>, <CQI2>}subframe 1, {...}subframe 2, ..., {...}subframe n The number of subframes n is configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ULLogging:DCHSpa
value: ResultData = driver.uplinkLogging.dchspa.read()
```

Return all results of the UL logging measurement on the E-DPCCH/DPCCH/HS-DPCCH. The results are returned as groups per measured subframe: <Reliability>, {<SFN>, <Slot>, <ETFCI>, <RSN>, <HappyBit>, <DPCCH1>, <DPCCH2>, <DPCCH3>, <ACKNACK1>, <CQI1>, <ACK-NACK2>, <CQI2>}subframe 1, {...}subframe 2, ..., {...}subframe n The number of subframes n is

configured via method RsCmwWcdmaSig.Configure.UplinkLogging.msFrames. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

7.17 Eagch

SCPI Commands

```
STOP:WCDMa:SIGNaling<Instance>:EAGCh
ABORt:WCDMa:SIGNaling<Instance>:EAGCh
INITiate:WCDMa:SIGNaling<Instance>:EAGCh
READ:WCDMa:SIGNaling<Instance>:EAGCh
FETCh:WCDMa:SIGNaling<Instance>:EAGCh
```

class Eagch

Eagch commands group definition. 9 total commands, 2 Sub-groups, 5 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Measured_Frames: int: Number of already measured HSUPA subframes Range: 1 to 1E+6
- Tot_Etfci_Events: int: Sum of all detected E-TFCI events Range: 1 to 1E+6
- Missed_Det: float: Number of missed expected E-TFCI detections Range: 1 to 1E+6
- Missed_Det_Prop: float: Missed detection probability (4_MissedDet / 3_TotETFCIEvents) Range: 0 % to 100 %
- Happy_Bits: int: Number of happy happy bits Range: 1 to 1E+6
- Etfci_Nr: List[int]: Expected E-TFCI value Range: 0 to 127
- Etfci_Events: List[int]: Number of detections Range: 0 to 1E+6

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:EAGCh
driver.eagch.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:EAGCh
driver.eagch.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EAGCh
value: ResultData = driver.eagch.fetch()
```

Return all single value results of the E-AGCH measurement. The results are returned as groups per most frequently detected E-TFCI values: <1_Reliability>, <2_MeasedFrames>, <3_TotETFCIEvents>, <4_MissedDet>, <5_MissedDetProb>, <6_HappyBits>, {<7_ETFCINr>, <8_ETFCIEvents>}1, {...}2, ..., {...}8 The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:EAGCh
driver.eagch.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:EAGCh
driver.eagch.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EAGCh
value: ResultData = driver.eagch.read()
```

Return all single value results of the E-AGCH measurement. The results are returned as groups per most frequently detected E-TFCI values: <1_Reliability>, <2_MeasedFrames>, <3_TotETFCIEvents>, <4_MissedDet>, <5_MissedDetProb>, <6_HappyBits>, {<7_ETFCINr>, <8_ETFCIEvents>}1, {...}2, ..., {...}8 The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:EAGCh
driver.eagch.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:EAGCh
driver.eagch.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

Cloning the Group

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

Subgroups

7.17.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:EAGCh:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EAGCh:STATe
value: enums.ResourceState = driver.eagch.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.eagch.state.clone()
```

Subgroups

7.17.1.1 All

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:EAGCh: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:WCDMa:SIGNaling<instance>:EAGCh:STATe:ALL
value: FetchStruct = driver.eagch.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.

7.17.2 Trace

class Trace

Trace 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.eagch.trace.clone()
```

Subgroups

7.17.2.1 General

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:EAGCh:TRACe:GENeral
FETCh:WCDMa:SIGNaling<Instance>:EAGCh:TRACe:GENeral
```

class General

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

class ResultData

Response structure. Fields:

- Reliablility: int: See 'Reliability Indicator'
- Events: List[int]: Detections for all E-TFCI values 0 to 127 from first to last (most recent) measured subframe Range: 0 to 1E+6

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EAGCh:TRACe:GENeral
value: ResultData = driver.eagch.trace.general.fetch()
```

Return the results of the detected E-TFCI values 0 to 127 for 'Measurement Type' = 'General Histogram'.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EAGCh:TRACe:GENeral
value: ResultData = driver.eagch.trace.general.read()
```

Return the results of the detected E-TFCI values 0 to 127 for 'Measurement Type' = 'General Histogram'.

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

7.18 Ehich

SCPI Commands

```
STOP:WCDMa:SIGNaling<Instance>:EHICh
ABORt:WCDMa:SIGNaling<Instance>:EHICh
INITiate:WCDMa:SIGNaling<Instance>:EHICh
```

class Ehich

Ehich commands group definition. 17 total commands, 4 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:EHICh
driver.ehich.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:EHICh
driver.ehich.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:EHICh
driver.ehich.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

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

(continues on next page)

(continued from previous page)

```
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:EHICh
driver.ehich.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:EHICh
driver.ehich.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:EHICh
driver.ehich.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

Cloning the Group

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

Subgroups

7.18.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:EHICh:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EHICh:STATe
value: enums.ResourceState = driver.ehich.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.ehich.state.clone()
```

Subgroups

7.18.1.1 All

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:EHICh: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:WCDMa:SIGNaling<instance>:EHICh:STATe:ALL
value: FetchStruct = driver.ehich.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.

7.18.2 Trace

class Trace

Trace 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.ehich.trace.clone()
```

Subgroups

7.18.2.1 MpThroughput

class MpThroughput
MpThroughput 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.ehich.trace.mpThroughput.clone()
```

Subgroups

7.18.2.1.1 Carrier

class Carrier
Carrier 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.ehich.trace.mpThroughput.carrier.clone()
```

Subgroups

7.18.2.1.1.1 Current

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:EHICh:TRACe:MPTHroughput:CARRier<Carrier>:CURRent
FETCh:WCDMa:SIGNaling<Instance>:EHICh:TRACe:MPTHroughput:CARRier<Carrier>:CURRent
```

class Current
Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EHICh:TRACe:MPTHroughput:CARRier
↪<carrier>:CURRent
value: List[float] = driver.ehich.trace.mpThroughput.carrier.current.fetch()
```

Return the results of the E-HICH traces per carrier. Maximum possible throughput is theoretical 'Current' throughput that would be reached within measured ETFCI if no CRC errors occurred. The number of results N depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI.

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

Global Repeated Capabilities: repcap.Carrier

return current: N throughput values, from first to last (most recent) measured subframe
Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EHICh:TRACe:MPTHroughput:CARRier<carrier>
↪:CURRent
value: List[float] = driver.ehich.trace.mpThroughput.carrier.current.read()
```

Return the results of the E-HICH traces per carrier. Maximum possible throughput is theoretical 'Current' throughput that would be reached within measured ETFCI if no CRC errors occurred. The number of results N depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI.

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

Global Repeated Capabilities: repcap.Carrier

return current: N throughput values, from first to last (most recent) measured subframe
Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.18.2.2 MeThroughput

class MeThroughput
MeThroughput 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.ehich.trace.meThroughput.clone()
```

Subgroups

7.18.2.2.1 Carrier

class Carrier
Carrier 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.ehich.trace.meThroughput.carrier.clone()
```

Subgroups

7.18.2.2.1.1 Current

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:EHICh:TRACe:METHroughput:CARRier<Carrier>:CURRent
FETCh:WCDMa:SIGNaling<Instance>:EHICh:TRACe:METHroughput:CARRier<Carrier>:CURRent
```

class Current

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EHICh:TRACe:METHroughput:CARRier
↪<carrier>:CURRent
value: List[float] = driver.ehich.trace.meThroughput.carrier.current.fetch()
```

Return the results of the E-HICH traces per carrier. Maximum expected throughput reachable if the UE sends at the maximum data rate (depends on the current settings) and no CRC errors occur. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI.

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

Global Repeated Capabilities: repcap.Carrier

return current: N throughput values, from first to last (most recent) measured subframe
Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EHICh:TRACe:METHroughput:CARRier<carrier>
↪:CURRent
value: List[float] = driver.ehich.trace.meThroughput.carrier.current.read()
```

Return the results of the E-HICH traces per carrier. Maximum expected throughput reachable if the UE sends at the maximum data rate (depends on the current settings) and no CRC errors occur. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI.

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

Global Repeated Capabilities: repcap.Carrier

return current: N throughput values, from first to last (most recent) measured subframe
Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

7.18.2.3 Throughput

class Throughput
Throughput commands group definition. 4 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.18.2.3.1 Carrier

class Carrier
Carrier 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.ehich.trace.throughput.carrier.clone()
```

Subgroups

7.18.2.3.1.1 Current

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:EHICh:TRACe:THRoughput:CARRier<Carrier>:CURRent
FETCh:WCDMa:SIGNaling<Instance>:EHICh:TRACe:THRoughput:CARRier<Carrier>:CURRent
```

class Current
Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EHICh:TRACe:THRoughput:CARRier<carrier>
↪:CURRent
value: List[float] = driver.ehich.trace.throughput.carrier.current.fetch()
```

Return the results of the E-HICH traces per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return current: No help available

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EHICh:TRACe:THRoughput:CARRier<carrier>
↪:CURRent
value: List[float] = driver.ehich.trace.throughput.carrier.current.read()
```

Return the results of the E-HICH traces per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return current: No help available

7.18.2.3.1.2 Average

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:EHICh:TRACe:THRoughput:CARRier<Carrier>:AVERage
FETCh:WCDMa:SIGNaling<Instance>:EHICh:TRACe:THRoughput:CARRier<Carrier>:AVERage
```

class Average

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

fetch() → List[float]

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EHICh:TRACe:THRoughput:CARRier<carrier>
↪:AVERage
value: List[float] = driver.ehich.trace.throughput.carrier.average.fetch()
```

Return the results of the E-HICH traces per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return average: No help available

read() → List[float]

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EHICh:TRACe:THRoughput:CARRier<carrier>
↪:AVERage
value: List[float] = driver.ehich.trace.throughput.carrier.average.read()
```

Return the results of the E-HICH traces per carrier. The number of results depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI. The results of the average and current traces can be retrieved.

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

Global Repeated Capabilities: repcap.Carrier

return average: No help available

7.18.3 Carrier

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:EHICh:CARRier<Carrier>
FETCh:WCDMa:SIGNaling<Instance>:EHICh:CARRier<Carrier>
```

class Carrier

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Meas_Frames: int: Number of already measured HSUPA subframes Range: 0 to 1E+6
- False_Rx: int: Number of transmissions that the UE received incorrectly Range: 0 to 1E+6
- Correct_Rx: int: Number of transmissions that the UE received correctly Range: 0 to 1E+6
- All_Valid_Rx: int: Number of transmissions that the UE received correctly or incorrectly For all three 'RX' results, the first new data block after a complete retransmission cycle is not counted as a test sample. Range: 0 to 1E+6
- False_Ratio: float: Ratio of 3_FalseRX to 5_AllValidRX Range: 0 % to 100 %, Unit: %
- Correct_Crc: int: Number of transmissions with correct CRC Range: 0 to 1E+6
- Error_Crc: int: Number of transmissions with incorrect CRC Range: 0 to 1E+6
- Bler: float: Block error rate resulting from CRC results Range: 0 % to 100 %, Unit: %
- Thrpt_Current: float: Current throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Thrpt_Max_Pos: float: Current throughput if there would be no CRC errors Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Thrpt_Max_Exp: float: Expected maximum reachable throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Thrpt_Average: float: Average throughput Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Thrpt_Maximum: float: Maximum throughput since the start of the measurement Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s
- Thrpt_Minimum: float: Minimum throughput since the start of the measurement Range: 0 bit/s to 100E+6 bit/s, Unit: bit/s

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EHICh:CARRier<carrier>
value: ResultData = driver.ehich.carrier.fetch()
```

Return all single value results of the E-HICH measurement per carrier. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

Global Repeated Capabilities: repcap.Carrier

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EHICh:CARRier<carrier>
value: ResultData = driver.ehich.carrier.read()
```

Return all single value results of the E-HICH measurement per carrier. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

Global Repeated Capabilities: repcap.Carrier

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

7.18.4 Throughput

class Throughput

Throughput 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.ehich.throughput.clone()
```

Subgroups

7.18.4.1 Total

SCPI Commands

```
READ:WCDMa:SIGNaling<Instance>:EHICh:THRoughput:TOTal
FETCh:WCDMa:SIGNaling<Instance>:EHICh:THRoughput:TOTal
```

class Total

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

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Current: float: N throughput values, from first to last (most recent) measured subframe
- Average: float: Average of all 'Current' values referenced to the last statistics cycle

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:EHICh:THRoughput:TOTal
value: ResultData = driver.ehich.throughput.total.fetch()
```

Return the results of the E-HICH traces over all carriers. The number of results N depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:EHICh:THRoughput:TOTal
value: ResultData = driver.ehich.throughput.total.read()
```

Return the results of the E-HICH traces over all carriers. The number of results N depends on the configured number of subframes to be measured per measurement cycle, see method RsCmwWcdmaSig.Configure.Ehich.mframes. One measurement result is returned per 100 subframes for 2 ms TTI and per 20 frames for 10 ms TTI.

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

7.19 Ergch

SCPI Commands

```
STOP:WCDMa:SIGNaling<Instance>:ERGCh
ABORt:WCDMa:SIGNaling<Instance>:ERGCh
INITiate:WCDMa:SIGNaling<Instance>:ERGCh
FETCh:WCDMa:SIGNaling<Instance>:ERGCh
READ:WCDMa:SIGNaling<Instance>:ERGCh
```

class Ergch

Ergch commands group definition. 7 total commands, 1 Sub-groups, 5 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See ‘Reliability Indicator’
- Meas_Frames: int: Number of already measured HSUPA subframes
- Happy_Happy_Bits: int: Number of detected happy happy bits
- Missed_Up: int: Number of relative grant values that the UE received in error during ‘Missed Up’ test
- Missed_Down: int: Number of relative grant values that the UE received in error during ‘Missed Down’ test
- Correct_Up: int: Number of relative grant values that the UE received correctly during ‘Missed Up’ test
- Correct_Down: int: Number of relative grant values that the UE received correctly during ‘Missed Down’ test
- All_Valid_Up: int: Sum of the missed and the correct events during ‘Missed Up’ test

- All_Valid_Down: int: Sum of the missed and the correct events during 'Missed Down' test
- Missed_Up_Ratio: float: 4_MissedUp events / 8_AllValidUp events
- Missed_Down_Ratio: float: 5_MissedDown events / 9_AllValidDown events
- Missed_Hold: int: Number of relative grant values that the UE received in error during 'Missed Hold' test
- Correct_Hold: int: Number of relative grant values that the UE received correctly during 'Missed Hold' test
- All_Valid_Hold: int: Sum of the missed and the correct events during 'Missed Hold' test
- Missed_Hold_Ratio: float: 12_MissedHold events / 14_AllValidHold events

abort() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:ERGCh
driver.ergch.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:WCDMa:SIGNaling<instance>:ERGCh
driver.ergch.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → ResultData

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ERGCh
value: ResultData = driver.ergch.fetch()
```

Return all single value results of the E-RGCH measurement. 'Missed Up', 'Missed Down' and 'Missed Hold' test refers to wizard settings, see 'Using the WCDMA Wizards'. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

initiate() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:ERGCh
driver.ergch.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:WCDMa:SIGNaling<instance>:ERGCh
driver.ergch.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

read() → ResultData

```
# SCPI: READ:WCDMa:SIGNaling<instance>:ERGCh
value: ResultData = driver.ergch.read()
```

Return all single value results of the E-RGCH measurement. 'Missed Up', 'Missed Down' and 'Missed Hold' test refers to wizard settings, see 'Using the WCDMA Wizards'. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

stop() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:ERGCh
driver.ergch.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:WCDMa:SIGNaling<instance>:ERGCh
driver.ergch.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

Cloning the Group

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

Subgroups

7.19.1 State

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ERGCh:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwWcdmaSig.enums.ResourceState

```
# SCPI: FETCh:WCDMa:SIGNaling<instance>:ERGCh:STATe
value: enums.ResourceState = driver.ergch.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.ergch.state.clone()
```

Subgroups

7.19.1.1 All

SCPI Commands

```
FETCh:WCDMa:SIGNaling<Instance>:ERGCh: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:WCDMa:SIGNaling<instance>:ERGCh:STATe:ALL
value: FetchStruct = driver.ergch.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.

INDEX

A

C

```
        73                                                                                                                  139
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:NCONfigure:GSM:EWCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:MRET
        74                                                                                                                  139
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:NCONfigure:TDSCdma:SIGNaling<Instance>:UL:PRACh:PREamble:SIGN
        75                                                                                                                  139
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:NCONfigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:SSIZ
        74                                                                                                                  139
CONFigure:WCDMa:SIGNaling<Instance>:UEReport:RCONfigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PREamble:SUBC
        71                                                                                                                  139
CONFigure:WCDMa:SIGNaling<Instance>:UL:CARRier:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:MODE,
        135                                                                                                                 149
CONFigure:WCDMa:SIGNaling<Instance>:UL:CARRier:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:MPEDch:STATe,
        135                                                                                                                 153
CONFigure:WCDMa:SIGNaling<Instance>:UL:CARRier:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:PATTern,
        136                                                                                                                 149
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:PEXecute,
        142                                                                                                                 154
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:PRECondition,
        146                                                                                                                 153
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:SET,
        146                                                                                                                 150
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:STATe,
        146                                                                                                                 149
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:TPOWer:OFFSet,
        147                                                                                                                 152
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPC:TPOWer:REFerenc
        147                                                                                                                 152
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:DHIE
        147                                                                                                                 155
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:PHDc
        147                                                                                                                 155
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:PHUP
        144                                                                                                                 155
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:TSEP
        145                                                                                                                 155
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:TSGH
        142                                                                                                                 155
CONFigure:WCDMa:SIGNaling<Instance>:UL:GFACtor:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PCONfig:TSSe
        142                                                                                                                 155
CONFigure:WCDMa:SIGNaling<Instance>:UL:MUEPower:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition
        133                                                                                                                 157
CONFigure:WCDMa:SIGNaling<Instance>:UL:OLPControl:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition
        137                                                                                                                 157
CONFigure:WCDMa:SIGNaling<Instance>:UL:OLPControl:CONFigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition
        137                                                                                                                 157
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:DCONfigure:WCDMa:SIGNaling<Instance>:UL:TPCSet:PRECondition
        138                                                                                                                 157
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:MCONfigure:WCDMa:SIGNaling<Instance>:UL:UEPClass:MANual,
        141                                                                                                                 134
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:MCONfigure:WCDMa:SIGNaling<Instance>:UL:UEPClass:REPorted,
        141                                                                                                                 134
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PCONfigure:WCDMa:SIGNaling<Instance>:ULLogging:MSFRames,
        139                                                                                                                 292
CONFigure:WCDMa:SIGNaling<Instance>:UL:PRACh:PCONfigure:WCDMa:SIGNaling<Instance>:ULLogging:REPetition,
```

F

I

P

R

S